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 giovedì 11 gennaio 2024

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 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 right, wrong
sem_t law, crime
int global = 0
```

```
void * truth (void * arg) {
    mutex_lock (&right)
    sem_wait (&law)
    sem_post (&crime)
    global = 1                                          /* statement A */
    mutex_unlock (&right)
    mutex_lock (&wrong)
    sem_post (&law)
    mutex_unlock (&wrong)
    global = 2                                          /* statement B */
    return (void *) 3
} /* end truth */
```

```
void * lie (void * arg) {
    mutex_lock (&wrong)
    sem_wait (&law)
    sem_wait (&crime)
    mutex_unlock (&wrong)
    mutex_lock (&right)
    sem_post (&law)
    global = 4                                          /* statement C */
    mutex_unlock (&right)
    return NULL
} /* end lie */
```

```
void main ( ) {
    pthread_t th_1, th_2
    sem_init (&law, 0, 1)
    sem_init (&crime, 0, 0)
    create (&th_2, NULL, lie, NULL)
    create (&th_1, NULL, truth, 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 – *truth*	**th_2 – *lie***
subito dopo stat. **A**	*ESISTE*	*ESISTE*
subito dopo stat. **B**	*ESISTE*	*PUÒ ESISTERE*
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**			
	right	*wrong*	*law*	*global*
subito dopo stat. **A**	*1*	*0 / 1*	*0*	*1*
subito dopo stat. **B**	*0 / 1*	*0 / 1*	*0 / 1*	*2 / 4*
subito dopo stat. **C**	*1*	*0*	*1*	*2 / 3 / 4*
subito dopo stat. **D**	*0 / 1*	*0 / 1*	*0 / 1*	*3 / 4*

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 – *truth***	**th_2 – *lie***	*global*
1	`lock wrong`	`wait law`	*1*
2	`wait law`	`wait crime`	*0*
3			

Nota: lo schema TH1: wait law TH2: lock right è irraggiungibile, pertanto non dà luogo a deadlock.

esercizio n. 2 – processi e nucleo

prima parte – moduli del SO

Un task **P** esegue la chiamata di sistema **wait (NULL)** e si sospende in attesa della terminazione di un task figlio **Q**, che è in stato di **attesa** a causa di richiesta di **lettura** di periferica. Subito prima che il task **P**, dopo essere passato in modo **S**, chiami il servizio di sistema di autosospensione, si verifica l'interruzione che risveglia il task **Q**. L'interruzione ha priorità massima e la condizione di *preemption* risulta falsa.
Nel sistema non ci sono altri task.

Si **mostrino** la sequenza di invocazioni di moduli di SO completa, cioè fino quando il task **Q** sarà tornato a eseguire il suo codice utente, il contenuto della **sPila** di **P** e i contenuti delle **strutture dati** indicate nella tabella, a sequenza di invocazioni completata.

task	modo	modulo
P	U	codice_utente
P	*U*	*>wait (P si mette in attesa della terminazione del task figlio Q)*
P	***U → S***	*>syscall: SYSCALL*
P	*S*	*>System_Call (la funzione dovrebbe chiamare il servizio **sys_wait,** ma arriva prima un'interruzione a priorità più alta)*
P	*S*	*>R_Int (Int. per evento da periferica per Q)*
P	*S*	*>wake_up (rimette Q in runqueue)*
P	*S*	*>check_preempt_curr< (no resched: la cond. di preemption è falsa)*
P	*S*	*wake_up<*
P	*S*	*R_Int: IRET< (torna a System_Call, che ora chiama il servizio sys_wait)*
P	*S*	*>sys_wait (P si mette in attesa)*
P	*S*	*>schedule*
P	*S*	*>pick_next_task< (seleziona il task Q, l'unico presente in coda RB)*
P → Q	***S***	*schedule: context_switch*
Q	*S*	*schedule<*
Q	*S*	*wait_event< (il task Q si era autosospeso su evento da periferica)*
Q	*S*	*sys_read< (il task Q aveva iniziato il servizio di lettura di I/O)*
Q	***S → U***	*System_Call: SYSRET<*
Q	*U*	*syscall<*
Q	*U*	*read< (il task Q aveva effettuato una richiesta di lettura di I/O)*
Q	*U*	*codice_utente*

X	*USP (top uStack_P durante syscall)*
	~~*ind. rientro a schedule (da pk_nt_task)*~~
3	*ind. rientro a sys_wait (da schedule)*
2	*ind. rientro a System_Call (da sys_wait)*
	~~*ind. rientro a wake_up (da ch_pr_cur)*~~
	~~*ind. rientro a R_Int (da wake_up)*~~
	~~*PSR (S) (System_Call è in modo S)*~~
	~~*ind. rientro a System_Call (da R_Int)*~~
1	*PSR (U) (syscall è in modo U)*
sBase_P	*ind. rientro a syscall (da System_Call)*

sStack_P – finale

strutture dati finali (da completare)	
SSP	*sBase_Q*
USP	*top uStack_Q durante syscall chiamata da read*
stato di **P** (nel descrittore)	*ATTESA, per primitiva wait, di attesa che il figlio Q termini*
stato di **Q** (nel descrittore)	*PRONTO, essendo task corrente è in **esecuzione***
campo sp di **P** (nel descrittore)	*X*

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	2	T2	50		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	T2	1	0,5	3	1	30	51
RB	T3	1	0,5	3	1	40	52
WAITING	T1	2				20	50

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

```
Events of task T1:  WAKEUP after 6.0
Events of task T3:  CLONE at 1.0;  EXIT at 1.5
```

Simulare l'evoluzione del sistema per **quattro eventi** riempiendo le seguenti tabelle, **in parte già precompilate** (per indicare le condizioni di rescheduling e altri calcoli richiesti, utilizzare le tabelle finali):

EVENTO 1	**TIME**	**TYPE**	**CONTEXT**	**RESCHED**		
	3	*Q_SCADE*	*T2*	*vero*		
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**	
	2	6	*2*	*T3*	*52*	
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	*T3*	*1*	*0,5*	*3*	*1*	*40*	*52*
RB	*T2*	*1*	*0,5*	*3*	*1*	*33*	*54*
WAITING	T1	2				20	50

EVENTO 2	**TIME**	**TYPE**	**CONTEXT**	**RESCHED**		
	4	*CLONE*	*T3*	*falso*		
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**	
	3	6	*3*	*T3*	*53*	
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	*T3*	*1*	*0,33*	*2*	*1*	*41*	*53*
RB	*T2*	*1*	*0,33*	*2*	*1*	*33*	*54*
	T4	*1*	*0,33*	*2*	*1*	*0*	*55*
WAITING	T1	2				20	50

EVENTO 3		TIME	TYPE	CONTEXT	RESCHED		
		4,5	EXIT	T3	vero		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	2	6	2	T2	53,5		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T2	1	0,5	3	1	33	54
RB	T4	1	0,5	3	1	0	55
WAITING	T1	2				20	50

EVENTO 4		TIME	TYPE	CONTEXT	RESCHED		
		6	WAKEUP	T2	vero		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	3	6	4	T1	55		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T1	2	0,50	3	0,5	20	52
RB	T4	1	0,25	1,5	1	0	55
	T2	1	0,25	1,5	1	34,5	55,5

Calcolo del VRT iniziale del **task T4** creato dalla **CLONE** *eseguita dal* ***task T3****:*

$T4.VRT\ (iniziale) = VMIN + T4.Q \times T4.VRTC = 53 + 2 \times 1 =$ ***55***

Valutazione della cond. di rescheduling alla **CLONE** *eseguita dal* ***task T3****:*

$T4.VRT + WGR \times T4.LC < T3.VRT \Rightarrow 55 + 1 \times 0{,}33 = 55{,}33 < 53 \Rightarrow$ ***falso***

Calcolo del VRT del **task T1** risvegliato dalla **WAKEUP** *eseguita dal* ***task T2****:*

$T1.VRT = max\ (T1.VRT,\ VMIN - LT / 2) = max\ (50,\ 55 - 6/2) = max\ (50,\ 52) =$ ***52***

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

$T1.VRT + WGR \times T1.LC < T2.VRT \Rightarrow 52 + 1 \times 0{,}50 = 52{,}50 < 55 \Rightarrow$ ***vero***

Nota: l'attesa del task T1 è lunga, e dato che VMIN al tempo assoluto 6 raggiunge il valore 55, il VRT assegnato a T1 al risveglio è pari a VMIN – LT / 2 = 55 – 3 = 52 (come da formula sopra); pertanto T1 viene schedulato come task corrente con VRT = 52; si noti che VMIN resterà a 55 anche se il task corrente T1 ha VRT inferiore; si veda la funzione task_tick, che a VMIN dà il valore massimo: VMIN = max (VMIN, min (T1.VRT, T4.VRT)) = max (55, min (52, 55)) = max (55, 52) = 55; VMIN manterrà tale valore fino a quando tutti i task in runqueue (compreso quello corrente) avranno assunto un VRT > 55, poi riprenderà a crescere.

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 tre processi: **P**, **Q** e **R**). **Si svolgano i tre eventi successivi**.

```
PROCESSO: P  ************************************************************
    VMA : C  000000400,  2 , R , P , M , <X,0>
          D  000000600,  4 , W , P , A , <-1,0>
          P  7FFFFFFFC,  3 , W , P , A , <-1,0>
    PT: <c0 :1  R>  <c1 :- ->  <d0 :7 W>  <d1 :s1 R>  <d2 :2 W>  <d3 :4 W>
        <p0 :6  W>  <p1 :5 R>  <p2 :- ->
    process P - NPV of PC and SP:  c0,  p0

PROCESSO: Q  ************************************************************
    VMA : C  000000400,  2 , R , P , M , <X,0>
          D  000000600,  4 , W , P , A , <-1,0>
          P  7FFFFFFFC,  3 , W , P , A , <-1,0>
    PT: <c0 :1  R>  <c1 :- ->  <d0 :- ->  <d1 :- ->  <d2 :- ->  <d3 :- ->
        <p0 :s0 W>  <p1 :- ->  <p2 :- ->
    process Q - NPV of PC and SP:  c0,  p0

PROCESSO: R  ************************************************************
    VMA : C  000000400,  2 , R , P , M , <X,0>
          D  000000600,  4 , W , P , A , <-1,0>
          P  7FFFFFFFC,  3 , W , P , A , <-1,0>
    PT: <c0 :1 R>    <c1 :- ->  <d0 :- ->  <d1 :s1 R>  <d2 :- ->  <d3 :- ->
        <p0 :3 D W>  <p1 :5 R>  <p2 :- ->
    process R - NPV of PC and SP:  c0,  p0

____MEMORIA FISICA____(pagine libere: 2)_____________________________
      00 : <ZP>                ||  01 : Pc0 / Qc0 / Rc0 / <X,0> ||
      02 : Pd2                 ||  03 : Rp0  D                  ||
      04 : Pd3                 ||  05 : Pp1 / Rp1               ||
      06 : Pp0                 ||  07 : Pd0                     ||
      08 : ----                ||  09 : ----                    ||

____STATO del TLB_____________________________________________________
      Pc0 : 01 -  0: 1:        ||  Pp0 : 06 -  1: 1:            ||
      Pd2 : 02 -  1: 1:        ||  Pp1 : 05 -  1: 0:            ||
      Pd0 : 07 -  1: 1:        ||  Pd3 : 04 -  1: 1:            ||

SWAP FILE:    Qp0,  Pd1 / Rd1, ----, ----, ----, ----

LRU ACTIVE:   PD3,  PD2,  PD0,  RP0,  RC0,  PP0,  PC0

LRU INACTIVE: rp1,  pp1,  qc0
```

PT INIZIALE del processo **P** **(qui già mostrata in tabella per aggiornare evento 1)**				
c0: 1 R	c1: - -	d0: 7 W	d1: s1 R	d2: 2 W
d3: 4 W	p0: 6 W	p1: 5 R	p2: - -	

MEMORIA FISICA INIZIALE (qui già mostrata in tabella per aggiornare evento 1)	
00: <ZP>	01: Pc0 / Qc0 / Rc0 / <X, 0>
02: Pd2	03: Rp0 D
04: Pd3	05: Pp1 / Rp1
06: Pp0	07: Pd0
08: -----	09: -----

SWAP FILE INIZIALE (qui già mostrato in tabella per aggiornare evento 1)	
s0: Qp0	s1: Pd1 / Rd1
s2: -----	s3: -----
s4: -----	s5: -----

evento 1: *read* (Pc0, Pp0) – 4 *kswapd*

le pagine Pc0 e Pp0 sono allocate in memoria fisca e vengono lette in memoria (per 5 volte); kswapd ha due funzioni: (1) verifica che free = 2 < maxfree = 3, pertanto attiva PFRA – Free:2, toReclaim:1; pertanto ***la pagina condivisa Pp1 / Rp1 (05) va in swap file (swap_out in slot s2)****, togliendo le pagine Pp1 e Rp1 da inactive; la tabella delle pagine del processo P viene aggiornata; (2) aggiorna le liste LRU per 4 volte (dunque si raggiunge la stabilità), tenendo conto che inizialmente tutte le pagine del processo P, tranne Pp1 / Rp1 che è stata deallocata, sono accedute (vedi TLB iniziale), e che successivamente solo le pagine Pc0 e Pp0 sono accedute (in lettura); pertanto le pagine Pd3, Pd2, Pd0, Rp0 e Rc0 vengono spostate da active a inactive senza referenza (N.B: non importa se Rp0, Rc0 siano inizialmente accedute o no, dato che in lista inactive seguiranno Pd3, Pd2, Pd0; quanto a Rp1, essendo già scesa in inactive, non è inizialmente acceduta), mentre Pp0 e Pc0 restano in active con referenza*

PT del processo **P** (compilare solo le quattro celle non oscurate)				
c0: 1 R	c1: - -	d0: 7 W	d1: *s1* R	d2: 2 W
d3: 4 W	p0: *6* W	**p1: ~~5R~~ *~~s2~~* R**	p2: - -	

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / Rc0 / <X, 0>
02: Pd2	03: *Rp0 D*
04: Pd3	05: ~~***Pp1 / Rp1***~~**-----**
06: *Pp0*	07: *Pd0*
08: -----	09: -----

SWAP FILE	
s0: Qp0	s1: Pd1 / Rd1
s2: ***Pp1 / Rp1***	s3: -----
s4: -----	s5: -----

LRU ACTIVE: ~~*PD3, PD2, PD0, RP0, RC0,*~~ *PP0, PC0*_____

LRU INACTIVE: ~~*rp1, pp1,*~~ *pd3, pd2, pd0, rp0, rc0, qc0,*_____

evento 2: *read* (Pp1)

la lettura di PP1 causa ***swap_in della pagina Pp1 condivisa con Rp1****, allocandola in pagina fisica 05; si applica il meccanismo di swap cache: la pagina condivisa Pp1 / Rp1 resta in swap file, la pagina Pp1 viene inserita nella tabella delle pagine del processo P e viene marcata in sola lettura (ossia viene predisposta per COW), poi la pagina Pp1 viene messa in testa active con referenza e la pagina Rp1 in coda inactive senza referenza*

PT del processo **P** (compilare solo le quattro celle non oscurate)				
c0: 1 R	c1: - -	d0: 7 W	d1: *s1* R	d2: 2 W
d3: 4 W	p0: *6* W	**p1: ~~*s2R*~~ *5* R**	p2: - -	

process P	NPV of **PC**: *c0*	NPV of **SP**: *p1*

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / Rc0 / <X, 0>
02: Pd2	03: *Rp0 D*
04: Pd3	05: ***Pp1 / Rp1***
06: *Pp0*	07: *Pd0*
08: -----	09: -----

SWAP FILE	
s0: Qp0	s1: Pd1 / Rd1
s2: *Pp1 / Rp1*	s3: -----
s4: -----	s5: -----

LRU ACTIVE: *PP1, PP0, PC0* _____

LRU INACTIVE: *pd3, pd2, pd0, rp0, rc0, qc0, rp1*

evento 3: *write* (Pd1)

la scrittura di Pd1 causa ***swap_in della pagina Pd1*** *condivisa con Rd1; lo swap_in a sua volta attiva PFRA perché il numero di pagine libere diventa < minfree = 2; PFRA - Required:1, Free:2, toReclaim:2;* ***le due pagine Rp0 e Pd0 (pagine fisiche 03 e 07) e vanno in swap file (swap_out in slot s3 e s4);*** *le liste LRU vengono aggiornate di conseguenza, togliendo Rp0 e Pd0 da inactive; la pagina condivisa Pd1 / Rd1 viene dapprima caricata in pagina fisica 3 e predisposta per COW, ma dato che va scritta viene subito sdoppiata, e la pagina Pd1 viene allocata in pagina fisica o7; la tabella delle pagine del processo P viene aggiornata di conseguenza; la pagina Rd1 viene messa in coda inactive senza referenza e la pagina Pd1 viene messa in testa active con referenza*

PT del processo **P** (compilare solo le quattro celle non oscurate)				
c0: 1 R	c1: - -	d0: ~~***7W s4 W***~~	d1: ~~***s1R 7 W***~~	d2: 2 W
d3: 4 W	p0: *6 W*	p1: *5 R*	p2: - -	

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / Rc0 / <X, 0>
02: Pd2	03: ~~***Rp0 D Pd1 / Rd1***~~
04: Pd3	05: *Pp1 / Rp1*
06: *Pp0*	07: ~~***Pd0 Pd1***~~
08: -----	09: -----

SWAP FILE	
s0: Qp0	s1: ~~***Pd1 / Rd1***~~
s2: *Pp1 / Rp1*	s3: ***Rp0***
s4: ***Pd0***	s5: -----

LRU ACTIVE: *PD1, PP1, PP0, PC0* _____

LRU INACTIVE: *pd3, pd2,* ~~*pd0, rp0,*~~ *rc0, qc0, rp1, rd1* _____

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 : M0 000030000,  2 , W , P , M , <G,2>
    (tutte le altre VMA sono omesse)
____MEMORIA FISICA____(pagine libere: 3)_____________________________________
      00 : <ZP>                      ||  01 : Pc2 / <X, 2>                ||
      02 : Pp0                       ||  03 : <G, 2>                      ||
      04 : Pm00                      ||  05 : ----                        ||
      06 : ----                      ||  07 : ----                        ||
    PT: <c0: - ->   <c1: - ->   <c2: 1 R>  <d0: - ->  <p0: 2 W>  <p1: - ->
        <m00: 4 W>  <m01: - ->
    process P - NPV of PC and SP:  c2,  p0
____STATO del TLB____________________________________________________________
      Pc2  : 01 -  0: 1:             ||  Pp0 : 02 -  1: 1:                ||
      Pm00 : 04 -  1: 1:             ||       -----                       ||
```

Il processo **P** è in esecuzione. **Si svolgano i sei eventi seguenti (da 1 a 6)**.

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.

MEMORIA FISICA INIZIALE (qui già mostrata in tabella per aggiornare evento 1)	
00: <ZP>	01: Pc2 / <X, 2>
02: Pp0	03: <G, 2>
04: Pm00	05: -----
06: -----	07: -----

PT INIZIALE del processo **P (qui già mostrata in tabella per aggiornare evento 3)**				
c0: - -	c1: - -	c2: 1 R	d0: - -	p0: 2 W
p1: - -	m00: 4 W			

evento 1: fd1 = *open* (F) – *write* (fd1, 7000)

il processo P apre il file F, e aggiorna f_pos e f_count; il processo P deve scrivere due pagine di file F, carica da disco le pagine <F, 0> e <F, 1>, e le alloca nelle pagine fisiche 5 e 6; poi le scrive in memoria e le marca dirty; per il file F, ci sono due letture da disco e nessuna scrittura su disco

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

processo/i	file	f_pos	f_count	numero pag. lette (da disco)	numero pag. scritte (su disco)
P	**F**	*7000*	*1*	*2*	*0*

evento 2: *close* (fd1)

*il processo P chiude il file F, e aggiorna f_pos e f_count; le pagine di file <F, 0> e <F, 1>, allocate nelle pagine fisiche 5 e 6, sono dirty, e dato che ora f_count vale 0, esse vengono scaricate su disco (si vedano le specifiche iniziali), tuttavia restano allocate nelle pagine fisiche 5 e 6 come pagine di disk cache, ma **non** più marcate dirty; per il file F, ci sono nessuna lettura da disco e due scritture su disco*

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

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
~~*P*~~	**F**	~~*7000*~~	~~*1*~~ ***0***	*2*	~~*0*~~ *2*

evento 3: *fork* (R) – *context switch* (R)

*il processo P si biforca e crea il processo figlio R; la pagina Pp0 top della pila di P viene sottoposta a COW, ma il numero di pagine libere scende sotto a minfree; si attiva PFRA – Required:1, Free:1, toReclaim:2; vengono liberate le due pagine fisiche di disk cache 3, ossia <G, 2>, e 5, ossia <F, 0>; la pagina **Pp0 viene allocata in pagina fisica 3** e marcata dirty in memoria, mentre il processo figlio R tiene come testa pila la pagina Rp0 in pagina fisica 2 e la marca dirty in memoria e nella tabella pagine di R (la pagina Rp0 verrà marcata dirty anche nel TLB); le altre pagine in memoria, ossia Pm00 e Pc2, vengono condivise con R; la tabella delle pagine del processo figlio R viene aggiornata e la pagina Rm00 viene marcata in sola lettura (ossia viene predisposta per COW poiché la VMA M0 è privata e scrivibile); poi si ha context switch e il processo R va in esecuzione; il TLB viene svuotato e rinnovato, registrando la PTE della pagina Rc2 di codice corrente (vedi NPV di PC di processo P in situazione iniziale), con flag acceduta ma non dirty, e la PTE della pagina Rp0 di testa pila, con flag dirty e acceduta; invece la pagina condivisa Pm00 / Rm00 viene marcata dirty in memoria perché figurava dirty nel TLB del processo padre P, che ora è stato svuotato e preparato per il processo figlio R (ma senza contenere la PTE della pagina Rm00 dato che il processo R non ha ancora acceduta tale pagina); ci sono nessuna lettura da disco e nessuna scrittura su disco*

PT del processo **R** figlio del processo **P** (compilare solo le tre celle non oscurate)				
c0: - -	c1: - -	c2: *1 R*	d0: - -	p0: *2 W* ***D***
p1: - -	m00: ***4 WR D***			

MEMORIA FISICA	
00: <ZP>	01: *Pc2* ***/ Rc2 /*** *<X, 2>*
02: ~~***Pp0***~~ ***Rp0 D***	03: ~~***<G, 2>***~~ ***Pp0 D***
04: *Pm00* ***/ Rm00 D***	05: ~~***<F, 0>***~~***-----***
06: *<F, 1>*	07: -----

STATO DEL TLB del processo **R** figlio del processo **P**	
Rc2: 01 – 0 : 1	*Rp0: 02 – 1 : 1*
-------	-------

evento 4: fd2 = *open* (H) – *write* (fd2, 4000)

il processo R apre il file H, e aggiorna f_pos e f_count; il processo R deve scrivere una pagina di file H, carica da disco la pagina <H, 0> e la alloca nella pagina fisica 5; poi la scrive in memoria e la marca dirty; per il file H, ci sono una lettura da disco e nessuna scrittura su disco

MEMORIA FISICA	
00: <ZP>	01: *Pc2 / Rc2 / <X, 2>*
02: *Rp0 D*	03: *Pp0 D*
04: *Pm00 / Rm00 D*	05: ***<H, 0> D***
06: *<F, 1>*	07: -----

processo/i	**file**	**f_pos**	**f_count**	**numero pag. lette**	**numero pag. scritte**
R	**H**	*4000*	*1*	*1*	*0*

evento 5: fd3 = *open* (F) – *write* (fd3, 8000)

il processo R apre nuovamente il file F, e aggiorna f_pos e f_count; il processo R deve leggere due pagine di file F e carica da disco la pagina <F, 0>, ma le pagine libere scendono sotto a minfree; si attiva PFRA – Required:1, Free:1, toReclaim:2; vengono liberate le pagine fisiche di disk cache 5, ossia <H, 0>, e 6, ossia <F, 1>; la pagina <H, 0> è dirty e viene scaricata su disco; poi il processo R carica da disco la pagina <F, 0> e la alloca in pagina fisica 5, e carica da disco (di nuovo) la pagina <F, 1> in pagina fisica 6; poi le scrive entrambe in memoria e le marca entrambe dirty; per il file H, ci sono nessuna lettura da disco e una scrittura su disco; per il file F, ci sono due letture da disco (che si sommano alle due precedenti) e nessuna scrittura su disco (si riportano le due precedenti)

MEMORIA FISICA	
00: <ZP>	01: *Pc2 / Rc2 / <X, 2>*
02: *Rp0 D*	03: *Pp0 D*
04: *Pm00 / Rm00 D*	05: ~~***<H, 0> D***~~ ***<F, 0> D***
06: ~~***<F, 1>***~~ ***<F, 1> D***	07: -----

processo/i	**file**	**f_pos**	**f_count**	**numero pag. lette**	**numero pag. scritte**
R	**H**	*4000*	*1*	*1*	~~*0*~~ *1*
R	**F**	*8000*	*1*	*2 + 2 = 4*	*2*

evento 6: *close* (fd2) – *close* (fd3)

il processo R chiude il file H (che non ha pagine presenti in memoria), e aggiorna f_pos e f_count (che scende a 0); il processo R chiude il file F, e aggiorna f_pos e f_count, che scende a 0; le pagine di file <F, 0> e <F, 1> sono in memoria e sono dirty, e dato che ora f_count vale 0, esse vengono scaricate su disco (si vedano le specifiche iniziali), tuttavia restano allocate nelle pagine fisiche 5 e 6 come pagine di disk cache, ma non più marcate dirty; per il file H, ci sono nessuna lettura da disco e nessuna scrittura su disco; per il file F, ci sono nessuna lettura da disco e due scritture su disco (che si sommano alle precedenti)

MEMORIA FISICA	
00: <ZP>	01: *Pc2 / Rc2 / <X, 2>*
02: *Rp0 D*	03: *Pp0 D*
04: *Pm00 / Rm00 D*	05: *<F, 0>* ~~D~~
06: *<F, 1>* ~~D~~	07: -----

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
~~R~~	**H**	~~4000~~	~~1~~ 0	1	1
~~R~~	**F**	~~8000~~	~~1~~ 0	4	~~2~~ 4

esercizio n. 4 – tabella delle pagine

Date le VMA di un processo, definire:

1. la scomposizione degli indirizzi virtuali dello NPV iniziale di ogni VMA 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 occupazione della Tabella delle Pagine (TP) e 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

Sono dati due casi (1 e 2) **da svolgere in successione**, e **una domanda finale**, come segue:

Caso 1) Si consideri che il processo P abbia le VMA seguenti:

VMA del processo **P**							
AREA	**NPV ini--------ziale**	**dimensione**	**R/W**	**P/S**	**M/A**	**nome file**	**offset**
C	0000 0040 0	16	R	P	M	X	0
K	0000 0060 0	2	R	P	M	X	3
S	0000 0060 2	4	W	P	M	X	4
D	0000 0060 6	2	W	P	A	-1	0
P	7FFF FFFF B	4	W	P	A	-1	0

Completare la tabella di scomposizione degli indirizzi virtuali:

AREA	**NPV iniziale**	**PGD**	**PUD**	**PMD**	**PT**
C	0000 0040 0	*0*	*0*	*2*	*0*
K	0000 0060 0	*0*	*0*	*3*	*0*
S	0000 0060 2	*0*	*0*	*3*	*2*
D	0000 0060 6	*0*	*0*	*3*	*6*
P	7FFF FFFF B	*255*	*511*	*511*	*507*

Indicare il numero di pagine necessarie per la Tabella delle Pagine (TP) del processo **P**:

# pag PGD	***1*** *(una sola per definizione)*
# pag PUD	***2*** *(il PGD contiene 2 PTE valide)*
# pag PMD	***2*** *(i PUD contengono 2 PTE valide)*
# pag PT	***3*** *(i PMD contengono 3 PTE valide)*
# pag totali	***8*** *(somma directory)*

Indicare i valori seguenti:

numero di pagine virtuali occupate dal processo	***28*** *(somma delle VMA)*
rapporto di occupazione	*8 / 28 =* ***0,286*** *(28,6%)*
dimensione massima del processo in pagine virtuali	*con la stessa dimensione di TP, il processo può crescere fino a 3 × 512 =* ***1536*** *pagine virtuali*
rapporto di occupazione con dimensione massima	*8 / 1536 =* ***0,0052*** *(0,52%)*

Caso 2) Si consideri che il processo P abbia creato una VMA **M0** di tipo Memory Mapped e una VMA di pila per il thread **T0**, le quali VMA si aggiungono alle VMA già viste nel caso (1), come segue:

VMA del processo **P**							
AREA	**NPV iniziale**	**dimensione**	**R/W**	**P/S**	**M/A**	**nome file**	**offset**
C	0000 0040 0	16	R	P	M	X	0
K	0000 0060 0	2	R	P	M	X	3
S	0000 0060 2	4	W	P	M	X	4
D	0000 0060 6	2	W	P	A	-1	0
M0	**0000 2000 0**	**4**	**W**	**S**	**M**	**G**	**1**
T0	**7FFF F77F E**	**2**	**W**	**P**	**A**	**-1**	**0**
P	7FFF FFFF B	4	W	P	A	-1	0

Aggiornare la tabella di scomposizione degli indirizzi virtuali, con le scomposizioni per le aree **M0** e **T0**:

AREA	**NPV iniziale**	**PGD**	**PUD**	**PMD**	**PT**
M0	**0000 2000 0**	*0*	*0*	*256*	*0*
T0	**7FFF F77F E**	*255*	*511*	*443*	*510*

Indicare il numero di pagine necessarie per la Tabella delle Pagine del processo **P** (tenendo conto di tutte le VMA del processo, caso 1 e caso 2)

# pag PGD	***1*** *(una sola per definizione)*
# pag PUD	***2*** *(il PGD contiene 2 PTE valide)*
# pag PMD	***2*** *(i PUD contengono 2 PTE valide)*
# pag PT	***5*** *(i PMD contengono 3 PTE valide da prima + 2 PTE valide per M0 e T0)*
# pag totali	***10*** *(somma directory)*

numero di pagine virtuali occupate dal processo	***34***
rapporto di occupazione	*10 / 34 = **0,294** (29,4%)*
dimensione massima del processo in pagine virtuali	*con la stessa dimensione di TP, il processo può crescere fino a 5 × 512 = **2560** pagine virtuali*
rapporto di occupazione con dimensione massima	*10 / 2560 = **0,004** (0,4%)*

Domanda finale: In quale dei due casi 1 e 2 il processo consuma una porzione di *spazio di memoria di sistema* (ossia pagine di kernelspace) proporzionalmente maggiore rispetto alla porzione di *spazio di memoria di utente* (ossia pagine di userspace) ? Motivare sinteticamente la risposta data.

Essendo la tabella delle pagine una struttura del kernel, le pagine che la compongono (PGD, PUD, PMD e PT) stanno in kernel space, mentre le pagine delle varie VMA C, K, S, D, P e M0, T0 sono in user space. I consumi di spazio di sistema rispetto allo spazio utente sono dunque espressi dai rapporti di occupazione calcolati nei due casi.

*Facendo riferimento ai rapporti di occupazione effettivi, si ha 0,294 (caso 2) > 0,286 (caso 1) cioè il consumo è **maggiore nel caso 2.***