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) _____

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**		
subito dopo stat. **B**		
subito dopo stat. **C**		
subito dopo stat. **D**		

Si completi la tabella qui sotto, **indicando i valori delle variabili globali** (sempre esistenti) nell'istante di tempo specificato da ciascuna condizione. Il **valore** della variabile va indicato così:

- intero, carattere, stringa, quando la variabile ha un valore definito; oppure X quando è indefinita
- se la variabile può avere due o più valori, li si riporti tutti quanti
- il semaforo può avere valore positivo o nullo (non valore negativo)
- si supponga che il mutex valga 1 se occupato, e valga 0 se libero

Si badi bene alla colonna "condizione": con "subito dopo statement X" si chiede il valore (o i valori) che la variabile ha tra lo statement X e lo statement immediatamente successivo del *thread* indicato.

condizione	**variabili globali**		
	mars	*venus*	*sun*
subito dopo stat. **B**			
subito dopo stat. **C**			
subito dopo stat. **D**			

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			
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			
	TGID	1	2			
P – fork	0	pronto	esec	pronto	NE	NE
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					
	9					
	10		pronto			esec

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

	11					
P – exit	12					

caso 2: sequenza finale qualora *TH_2 termini*

	11					
	12					
	13					
P – exit	14					

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**

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  7FFFFFF9,  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  7FFFFFF9,  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*

PT del processo: **P**				
p0:	p1:	p2:	p3:	p4:

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

PT del processo: R				
p0:	p1:	p2:	p3:	p4:

MEMORIA FISICA	
00: <ZP>	01: Pc1 / Rc1 / <XX, 1>
02:	03:
04:	05:
06:	07:
08:	09:

SWAP FILE	
s0: Rp0	s1: Pp1 / Rp1
s2:	s3:
s4:	s5:

LRU ACTIVE: _____

LRU INACTIVE: _____

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)

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							
M1							

PT del processo: **P**				
p2:	p3:	p4:	p5: - -	m00:
m01:	m02:	m10:	m11:	m12:

MEMORIA FISICA	
00: <ZP>	01: Pc1 / Rc1 / <XX, 1>
02:	03:
04:	05:
06:	07:
08:	09:

SWAP FILE	
s0: Rp0	s1: Pp1 / Rp1
s2:	s3:
s4:	s5:

LRU ACTIVE: _____

LRU INACTIVE: _____

(continua sulla prossima pagina)

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

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

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

LRU ACTIVE: _____

LRU INACTIVE: _____

evento 4: *write* (Pm11)

PT del processo: **P**				
m01:	m02:	m10:	m11:	m12:

MEMORIA FISICA	
00: <ZP>	01: Pc1 / Rc1 / <XX, 1>
02:	03:
04:	05:
06:	07:
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)

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

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

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

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

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

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

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

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

evento 4: *write* (fd1, 3000)

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

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

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

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

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				
D	000000608				
M0	000010000	0	0	128	0
M1	000020000				
M2	000030000				
M3	000050000	0	1	128	0
T2	7FFFF77F8				
T1	7FFFF77FB				
T0	7FFFF77FE	255	511	443	510
P	7FFFFFFFC				

2. Numero di pagine necessarie:

pag PGD: # pag PUD:

pag PMD: # pag PT:

pag totali:

3. Numero di pagine virtuali occupate dal processo:

4. Rapporto di occupazione:

5. Dimensione massima del processo in pagine virtuali:

6. Rapporto di occupazione con dimensione massima:

spazio libero per brutta copia o continuazione