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 – giovedì 17 giugno 2021

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 (6 punti) _____

esercizio 3 (4 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 deep, shallow
sem_t water
int global = 0
```

```
void * boat (void * arg) {
   mutex_lock (&deep)
   sem_post (&water)
   global = 1                                          /* statement A */
   mutex_unlock (&deep)
   mutex_lock (&shallow)
   global = 2
   sem_wait(&water)
   mutex_unlock (&shallow)
   sem_post (&water)
   return NULL
} /* end boat */
```

```
void * ship (void * arg) {
   mutex_lock (&deep)
   sem_wait (&water)
   mutex_lock (&shallow)
   global = 3                                          /* statement B */
   mutex_unlock (&shallow)
   sem_post(&water)
   mutex_unlock (&deep)                                /* statement C */
   sem_wait (&water)
   return (void *) 4
} /* end ship */
```

```
void main ( ) {
   pthread_t th_1, th_2
   sem_init (&water, 0, 0)
   create (&th_1, NULL, boat, NULL)
   create (&th_2, NULL, ship, 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 – *boat*	**th_2 – *ship***
subito dopo stat. **A**	*ESISTE*	*PUÒ ESISTERE*
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*	*0*	*1*	*1*
subito dopo stat. **B**	*1*	*1*	*0*	*3*
subito dopo stat. **C**	*0*	*0 / 1*	*0 / 1*	*2 / 3*
subito dopo stat. **D**	*0*	*0 / 1*	*0*	*2 / 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 – *boat***	**th_2 – *ship***	*global*
1	`lock deep`	`prima wait water`	*0*
2	`wait water`	`lock shallow`	*2*
3	`wait water`	`-`	*2 / 4*

esercizio n. 2 – processi e nucleo

prima parte – gestione dei processi

```
// programma schiaccia_tre.c
// dichiarazione e inizializzazione dei mutex presenti nel codice
// dichiarazione dei semafori presenti nel codice
```

`void * player1 (void * arg) {`	`void * player2 (void * arg) {`
`  sem_wait (&player1_has_ball)`	`  sem_wait (&player2_has_ball)`
`  sem_post (&player2_has_ball)`	`  sem_post (&player3_has_ball)`
`  write (stdout, "One", 3)`	`  write (stdout, "Two", 3)`
`} // end player1`	`} // end player2`
`void * player3 (void * arg) {`	`void * player4 (void * arg) {`
`  sem_wait (&player3_has_ball)`	`  mutex_lock (&trick)`
`  mutex_lock (&trick)`	`  sem_wait (&player4_has_ball)`
`  sem_post (&player4_has_ball)`	`  write ("ouch...", 7)`
`  write (stdout, "THREE!", 6)`	`  mutex_unlock (&trick)`
`  mutex_unlock (&trick)`	`} // end palyer4`
`} // end player3`	

```
main ( ) {  // codice eseguito da Q
    pthread_t p1, p2, p3, p4
    sem_init (&player1_has_ball , 0, 0)
    sem_init (&player2_has_ball , 0, 0)
    sem_init (&player3_has_ball , 0, 0)
    sem_init (&player4_has_ball , 0, 0)
    create (&p4, NULL, player4, NULL)
    create (&p3, NULL, player3, NULL)
    create (&p1, NULL, player1, NULL)
    create (&p2, NULL, player2, NULL)
    write (stdout, "Ready, set, go!", 15)
    sem_post (&player1_has_ball)
    join (p3, NULL)
    join (p2, NULL)
    join (p1, NULL)
    join (p4, NULL)
    exit (0)
} // main
```

Un processo ***Q*** esegue il programma **schiaccia_tre** e crea ordinatamente i thread ***p4***, ***p3***, ***p1*** e ***p2***.

Si simuli l'esecuzione dei processi completando tutte le righe presenti nella tabella così come risulta dal codice dato, dallo stato iniziale, dagli eventi indicati, e nell'ipotesi che il processo ***Q*** **non abbia creato nessun thread**. Si completi la tabella riportando quanto segue:

- ⟨ *PID*, *TGID* ⟩ di ciascun processo che viene creato
- ⟨ *identificativo del processo-chiamata di sistema / libreria* ⟩ nella prima colonna, dove necessario e in funzione del codice proposto
- 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 (numero di colonne non significativo)

identificativo simbolico del processo		**IDLE**	**Q**	**p4**	**p3**	**p1**	**p2**	
	PID	**1**	2	*3*	*4*	*5*	*6*	
evento oppure *processo-chiamata*	*TGID*	**1**	2	*2*	*2*	*2*	*2*	
Q – create (p4)	0	**pronto**	**esec**	**pronto**	*NE*	*NE*	*NE*	
Q – create (p3)	*1*	*pronto*	**esec**	*pronto*	*pronto*	*NE*	*NE*	
Q – create (p1)	*2*	*pronto*	**esec**	*pronto*	*pronto*	*pronto*	*NE*	
Q – create (p2)	*3*	*pronto*	**esec**	*pronto*	*pronto*	*pronto*	*pronto*	
Q – write (stdout)	4	*pronto*	*A (write)*	*esec*	*pronto*	*pronto*	*pronto*	
p4 – mutex_lock (trick)	5	*pronto*	*A (write)*	*esec*	*pronto*	*pronto*	*pronto*	
p4 – sem_wait (player4_has_ball)	6	*pronto*	*A (write)*	*A (sem)*	*esec*	*pronto*	*pronto*	
15 interrupt da STD_OUT tutti i byte trasferiti	7	*pronto*	*esec*	*A(sem)*	*pronto*	*pronto*	*pronto*	
Q – sem_post (player1_has_ball)	8	**pronto**	**esec**	**A (sem)**	**pronto**	**pronto**	**pronto**	
interrupt da real-time clock e scadenza del quanto di tempo	9	*pronto*	*pronto*	*A (sem)*	*pronto*	*esec*	*pronto*	
p1 – sem_wait (player1_has_ball)	10	*pronto*	*pronto*	*A (sem)*	*pronto*	*esec*	*pronto*	
p1 – sem_post (player2_has_ball)	11	*pronto*	*pronto*	*A (sem)*	*pronto*	*esec*	*pronto*	
p1 – write (stdout)	12	*pronto*	*pronto*	*A (sem)*	*pronto*	*A(write)*	*esec*	
p2 – sem_wait (player2_has_ball)	13	*pronto*	*pronto*	*A (sem)*	*pronto*	*A (write)*	*esec*	
p2 – sem_post (player3_has_ball)	14	*pronto*	*pronto*	*A (sem)*	*pronto*	*A (write)*	*esec*	
p2 – write (stdout)	15	*pronto*	*pronto*	*A (sem)*	*pronto*	*A (write)*	*A (write)*	
p3 – sem_wait (player3_has_ball)	16	*pronto*	*pronto*	*A (sem)*	*esec*	*A (write)*	*A (write)*	

seconda parte – scheduling

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**		
	3	6	6	t1	100		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	t1	2	0,33	2	0,50	10	100
RB	t2	3	0,50	3	0,33	20	101
	t3	1	0,17	1	1,00	30	101,50

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

```
Events of task t1:  EXIT at 4.0;
Events of task t2:  WAIT at 1.0;   WAKEUP after 1.5;
```

Simulare l'evoluzione del sistema per **quattro eventi** riempiendo le seguenti tabelle (per indicare la condizione di rescheduling della *clone*, e altri calcoli eventualmente richiesti, utilizzare le tabelle finali):

EVENTO 1	**TIME**	**TYPE**	**CONTEXT**	**RESCHED**			
	2	*Q_scade*	*t1*	*vero*			
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
	3	6	*6*	*t2*	*101*		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	*t2*	*3*	*0,5*	*3*	*0,33*	*20*	*101*
RB	*t1*	*2*	*0,33*	*2*	*0,50*	*12*	*101*
	t3	*1*	*0,17*	*1*	*1,00*	*30*	*101,50*
WAITING							

EVENTO 2	**TIME**	**TYPE**	**CONTEXT**	**RESCHED**			
	3	*WAIT*	*t2*	*vero*			
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
	2	6	*3*	*t1*	*101*		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	*t1*	*2*	*0,67*	*4*	*0,50*	*12*	*101*
RB	*t3*	*1*	*0,33*	*2*	*1,00*	*30*	*101,50*
WAITING	*t2*	*3*				*21*	*101,33*

EVENTO 3		TIME	TYPE	CONTEXT	RESCHED		
		4,5	*WAKEUP*	*t1*	*falso*		
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
	3	6	*6*	*t1*	*101,50*		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	*t1*	*2*	*0,33*	*2*	*0,50*	*13*	*101,75*
RB	*t2*	*3*	*0,50*	*3*	*0,33*	*21*	*101,33*
	t3	*1*	*0,17*	*1*	*1,00*	*30*	*101,50*
WAITING							

EVENTO 4		TIME	TYPE	CONTEXT	RESCHED		
		5	*EXIT*	*t1*	*vero*		
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
	2	6	*4*	*t3*	*101,50*		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	*t2*	*3*	*0,75*	*4,50*	*0,33*	*21*	*101,33*
RB	*t3*	*1*	*0,25*	*1,50*	*1,00*	*30*	*101,50*
WAITING							

Valutazione della *condizione di rescheduling* alla **WAKEUP**:

$tw.vrt + WGR \times tw.LC = 101,33 + 1,00 \times 0,50 = 101,83 < curr.vrt = 101,75 \Rightarrow$ *falso*

esercizio n. 3 – memoria

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

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

situazione iniziale (esiste un solo processo P)

```
PROCESSO: P  ********************************************************
    VMA : C  000000400,  2 , R  ,  P  ,  M  , <X, 0>
          S  000000600,  2 , W  ,  P  ,  M  , <X, 2>
          P  7FFFFFFC,  3 , W  ,  P  ,  A  , <-1,0>
    PT: <c0 :1 R>  <c1 :-  ->  <s0 :- ->  <s1 :- ->  <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 : ----                  ||

SWAP FILE:     ----, ----, ----, ----, ----, ----,
LRU ACTIVE:    PP0, PC0
LRU INACTIVE:
```

evento 1: *read* (Pc0, Ps0), *write* (Pp0, Pp1)

alloca pagine 3 e 4 per s0 e p1, e inserisce s0 e p1 in lista attiva

PT del processo: **P**				
c0: *1* *R*	c1: - -	s0: *3* *R*	s1: - -	p0: *2* *W*
p1: *4* *W*	p2: - -			

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

MEMORIA FISICA	
00: <ZP>	01: Pc0 / <X, 0>
02: Pp0	03: *Ps0 / <X, 2>*
04: *Pp1*	05: ----
06: ----	07: ----

LRU ACTIVE: *PP1, PS0, PP0, PC0*_____

LRU INACTIVE: _____

evento 2: *read* (Pc0, Ps1) – *write* (Pp1) – 4 *kswapd*

alloca pagina 5 per s1 e aggiorna liste

PT del processo: **P**				
c0: *1* *R*	c1: - -	s0: *3* *R*	s1: *5* *R*	p0: *2* *W*
p1: *4* *W*	p2: - -			

MEMORIA FISICA	
00: <ZP>	01: *Pc0 / <X, 0>*
02: *Pp0*	03: *Ps0 / <X, 2>*
04: *Pp1*	05: *Ps1 / <X, 3>*
06: ----	07: ----

LRU ACTIVE: *PS1, PP1, PC0* _____

LRU INACTIVE: *ps0, pp0*_____

evento 3: ***mmap*** **(0x000000002000000, 2, W, S, M, "F", 0)** **VMA M0**

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	*0000 0200 0*	*2*	*W*	*S*	*M*	*F*	*0*

evento 4: ***read*** **(Pc0, Pm00)**

alloca pagina 6 per m00 e inserisce m00 in lista attiva

PT del processo: **P**				
p1: *4 W*	p2: - -	m00: *6 W*	m01: - -	

MEMORIA FISICA	
00: <ZP>	01: *Pc0 / <X, 0>*
02: *Pp0*	03: *Ps0 / <X, 2>*
04: *Pp1*	05: *Ps1 / <X, 3>*
06: *Pm00 / <F, 0>*	07: *----*

LRU ACTIVE: *PM00, PS1, PP1, PC0* _____

LRU INACTIVE: *ps0, pp0*_____

evento 5: ***clone*** **(R, c1)**

clona proc P e crea thread R, crea area pila T0, COW per t00, PFRA per liberare p0 e s0, scarica p0 in swap

PT del processo: **P**				
t00: *2 W*	t01: - -	m00: *6 W*	m01: - -	

process R	NPV of **PC**: *c1*	NPV of **SP**: *t00*

MEMORIA FISICA	
00: <ZP>	01: *PRc0 / <X, 0>*
02: *PRt00*	03: *----*
04: *PRp1*	05: *PRs1 / <X, 3>*
06: *PRm00 / <F, 0>*	07: *----*

SWAP FILE	
s0: *PRp0*	s1:

LRU ACTIVE: *PRT00, PRM00, PRS1, PRP1, PRC0* _____

LRU INACTIVE: _____

evento 6: 4 ***kswapd***

evento 7: ***context_switch*** **(R),** ***read*** **(Rm01)**

aggiorna liste (va tutto in inactive), commutazione a R e refresh TLB, t00 è dirty per P, alloca pagina 3 per c1 (inizio codice di R), COW per m01, PFRA per liberare c0 e p1, scarica p1 in swap

MEMORIA FISICA	
00: <ZP>	01: *PRm01 / <F, 1>*
02: *PRt00 D*	03: *PRc1 / <X, 1>*

04: *----*	05: *PRs1 / <X, 3>*
06: *PRm00 / <F, 0>*	07: *----*

SWAP FILE	
s0: *PRp0*	s1: *PRp1*

esercizio n. 4 – file system

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

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

Si consideri la seguente **situazione iniziale**:

```
____MEMORIA FISICA____(pagine libere: 5)_____________________________
      00 : <ZP>                  ||  01 : Pc0 / <X, 0>          ||
      02 : Pp0                   ||  03 : ----                  ||
      04 : ----                  ||  05 : ----                  ||
      06 : ----                  ||  07 : ----                  ||
```

Per ciascuno dei seguenti eventi compilare le tabelle richieste con i dati relativi al contenuto della memoria fisica, delle variabili del FS relative ai file **F** e **G** al numero di accessi a disco effettuati in lettura e in scrittura.

Il processo **P** è in esecuzione.

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 che la primitiva *close* scrive le pagine dirty di un file solo se *f_count* diventa = 0.

eventi 1 e 2: fd1 = *open* (F), *read* (fd1, 13000)

apre file e carica le pagine F0, F1, F2 e F3 nelle pagine 3, 4, 5, e 6 in memoria

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

nome file	f_pos	f_count	numero pag. lette	numero pag. scritte
F	13000	1	4	0

evento 3: *write* (fd1, 5000)

PFRA - Required:1, Free:1, To Reclaim:2; libera le pagine 3 e 4, carica la pagina F4 in pagina 3, scrive le pagine 3 e 6 in memoria

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

nome file	f_pos	f_count	numero pag. lette	numero pag. scritte
F	18000	1	5	0

eventi 4 e 5: *fork* (Q), *context switch* (Q)

crea processo figlio Q e assegna a Q la pagina pila p0, alloca e copia la pagina pila p0 del processo padre P in pagina 4 in memoria

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

nome file	f_pos	f_count	numero pag. lette	numero pag. scritte
F	*18000*	*2*	*5*	*0*

eventi 6 e 7: fd2 = *open* (G), *write* (fd2, 6000)

PFRA - Required:1, Free:1, To Reclaim:2; libera le pagine 3 e 5, scarica la pagina 3 su pagina F4, carica le pagine file G0 e G1 in pagine 3 e 5, scrive in pagine 3 e 5 in mem

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

nome file	f_pos	f_count	numero pag. lette	numero pag. scritte
F	*18000*	*2*	*5*	*1*
G	*6000*	*1*	*2*	*0*

evento 8: *write* (fd1, 3000)

PFRA - Required:1, Free:1, To Reclaim:2; libera le pagine 3 e 5 e le scarica su pagine G0 e G1, ricarica la pagina F4 e carica la pagina F5 in pagine 3 e 5, scrive le pagine 3 e 5 in memoria

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

nome file	f_pos	f_count	numero pag. lette	numero pag. scritte

F	*21000*	*2*	*7*	*1*
G	*6000*	*1*	*2*	*2*

spazio libero per brutta copia o continuazione