



Politecnico di Milano

Dipartimento di Elettronica, Informazione e Bioingegneria

prof. Luca Breveglieri

prof. Gerardo Pelosi

prof.ssa Donatella Sciuto

prof.ssa Cristina Silvano

AXO – Architettura dei Calcolatori e Sistemi Operativi

SECONDA PARTE – giovedì 29 giugno 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 (4 punti) _____

esercizio 4 (3 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 line
sem_t dot, bar
int global = 0
```

```
void * over (void * arg) {
    sem_wait (&dot)
    mutex_lock (&line)
    sem_post (&dot)
    global = 1
    mutex_unlock (&line)
```

```
    global = 2                                     /* statement A */
```

```
    mutex_lock (&line)
    sem_wait (&bar)
    mutex_unlock (&line)
    return (void *) 3
```

```
} /* end over */
```

```
void * under (void * arg) {
    mutex_lock (&line)
    sem_wait (&dot)
    mutex_unlock (&line)
    global = 4
```

```
    sem_post (&bar)                                /* statement B */
```

```
    sem_wait (&bar)
    mutex_lock (&line)
    global = 5
```

```
    sem_post (&bar)                                /* statement C */
```

```
    mutex_unlock (&line)
    return NULL
```

```
} /* end under */
```

```
void main ( ) {
    pthread_t th_1, th_2
    sem_init (&dot, 0, 1)
    sem_init (&bar, 0, 0)
    create (&th_1, NULL, over, NULL)
    create (&th_2, NULL, under, 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	<i>thread</i>	
	th_1 – over	th_2 – under
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		
	<i>line</i>	<i>dot</i>	<i>bar</i>
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 **TRE 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* (numero di righe non significativo):

caso	th_1 – over	th_2 – under	<i>global</i>
1			
2			
3			
4			

esercizio n. 2 – processi e nucleo

prima parte – gestione dei processi

// programma main.c	
sem_t sem	
char *strG = "Hello World!"	
char *strF = strG + 12	
void * day (void * arg) {	void * night (void * arg) {
if (strG < strF - 1) {	sem_wait (&sem)
sem_wait (&sem)	strG = strG + 6
strG = strG + 6	if (strG > strF - 1) {
write (stderr, strG, 6)	write (stderr, strG, 6)
} // end if	} // end if
return NULL	return NULL
} // end	} // end
int main () { // codice eseguito da P	
pid_t pidP, pidQ	
sem_init (&sem, 0, 0)	
pidQ = fork ()	
if (pidQ == 0) {	// codice eseguito da Q
write (stdout, strG, 11)	
exit (1)	
} else {	// codice eseguito da P
pthread_t TH_1, TH_2	
pthread_create (&TH_1, NULL, day , NULL)	
pthread_create (&TH_2, NULL, night , NULL)	
pidP = wait (NULL)	
sem_post (&sem)	
pthread_join (TH_1, NULL)	
sem_post (&sem)	
pthread_join (TH_2, NULL)	
exit (0)	
} // 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			
	TGID	1	2			
P – fork	0	pronto	esec	pronto	NE	NE
	1					
	2					
	3					
	4					
	5					
	6					
	7	pronto		esec		
	8					
	9				esec	
	10					
interrupt da stderr, sei caratteri inviati	11					
	12					
	13					
	14					
	15					
	16					
	17					

seconda parte – scheduler

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	5	T1	100		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T1	2	0,4	2,4	0,5	10	100
RB	T2	1	0,2	1,2	1	30	101
	T3	2	0,4	2,4	0,5	20	101,5

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

Events of task t1: WAIT at 1.0; WAKEUP after 1.5

Events of task t3: EXIT 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		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
		6					
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT							
RB							
WAITING							

EVENTO 2		TIME	TYPE	CONTEXT	RESCHED		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
		6					
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT							
RB							
WAITING							

EVENTO 3		TIME	TYPE	CONTEXT	RESCHED		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
		6					
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT							
RB							
WAITING							

EVENTO 4		TIME	TYPE	CONTEXT	RESCHED		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
		6					
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT							
RB							
WAITING							

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

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

esercizio n. 3 – memoria virtuale

È dato un sistema di memoria caratterizzato dai seguenti parametri generali (**ATTENZIONE a MAXFREE**):

MAXFREE = 4

MINFREE = 2

Situazione iniziale (esistono due processi P e Q)

PROCESSO: P *****

VMA : C 000000400, 2, R, P, M, <X, 0>
S 000000600, 3, W, P, M, <X, 2>
D 000000603, 4, W, P, A, <-1, 0>
P 7FFFFFFFB, 4, W, P, A, <-1, 0>
PT: <c0 :- -> <c1 :1 R> <s0 :- -> <s1 :5 R> <s2 :- -> <d0 :- ->
<d1 :s0 W> <d2 :3 W> <d3 :- -> <p0 :6 W> <p1 :- -> <p2 :s2 R>
<p3 :- ->
process P - NPV of PC and SP: c1, p0

PROCESSO: Q ***** (non di interesse) *****

MEMORIA FISICA (pagine libere: 4)

00 : <ZP>	01 : Pc1 / Qc1 / <X, 1>
02 : Qp0 D	03 : Pd2
04 : ----	05 : Ps1 / Qs1 / <X, 3>
06 : Pp0	07 : ----
08 : ----	09 : ----

STATO del TLB

Pc1 : 01 - 0: 1:	Pp0 : 06 - 1: 0:
Pd2 : 03 - 1: 0:	-----
Ps1 : 05 - 0: 0:	-----
-----	-----

SWAP FILE: Pd1, Qd1, Pp2 / Qp2, ----, ----, ----

LRU ACTIVE: PC1

LRU INACTIVE: pd2, ps1, pp0, qs1, qp0, qc1

evento 1: read (Pd0, Ps2) – write (Pd1)

MEMORIA FISICA	
00: <ZP>	01: Pc1 / Qc1 / <X, 1>
02:	03:
04:	05:
06:	07:
08:	09:

SWAP FILE	
s0:	s1:
s2:	s3:

LRU active: _____

LRU inactive: _____

evento 2: write (Pp2)

MEMORIA FISICA	
00: <ZP>	01: Pc1 / Qc1 / <X, 1>
02:	03:
04:	05:
06:	07:
08:	09:

SWAP FILE	
s0:	s1:
s2:	s3:

LRU active: _____

LRU inactive: _____

evento 3: read (Pc1) – write (Pd0) – 2 kswapd

MEMORIA FISICA	
00: <ZP>	01: Pc1 / Qc1 / <X, 1>
02:	03:
04:	05:
06:	07:
08:	09:

SWAP FILE	
s0:	s1:
s2:	s3:
s4:	s5:

LRU active: _____

LRU inactive: _____

esercizio n. 4 – file system

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

MAXFREE = 3 MINFREE = 2

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

È in esecuzione il processo **P**. La pagina in cima alla pila è **Pp0**.

ATTENZIONE: il numero di pagine lette o scritte di un file è cumulativo, quindi è la somma delle pagine lette o scritte su quel file da tutti gli eventi precedenti oltre a quello considerato.

eventi 1 e 2: *fd = open (F) read (fd, 8100)*

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

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F				

eventi 3-5: *fork (R) lseek (fd, -4500) write (fd, 500)*

MEMORIA FISICA	
00: <ZP>	01:
02:	03:
04:	05:
06:	07:

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F				

eventi 6-9: *fd1 = open (G) write (fd1, 4000) close (fd) close (fd1)*

MEMORIA FISICA	
00: <ZP>	01:
02:	03:
04:	05:
06:	07:

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F				
file G				

eventi 10 e 11: *context switch (R) write (fd, 100)*

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F				
file G				

spazio libero per brutta copia o continuazione