



Politecnico di Milano

Dipartimento di Elettronica, Informazione e Bioingegneria

prof.ssa Anna Antola

prof. Luca Breveglieri

prof. Roberto Negrini

prof. Giuseppe Pelagatti

prof.ssa Donatella Sciuto

prof.ssa Cristina Silvano

AXO – Architettura dei Calcolatori e Sistemi Operativi

SECONDA PARTE di 7 luglio 2017

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 (6 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):

```
pthread_mutex_t particle
sem_t force
int global = 0
```

```
void * plus (void * arg) {
    sem_wait (&force)
```

```
    sem_post (&force)                                /* statement A */
```

```
    pthread_mutex_lock (&particle)
```

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

```
    pthread_mutex_unlock (&particle)
```

```
    return 1
```

```
} /* end plus */
```

```
void * minus (void * arg) {
```

```
    global = 2                                        /* statement C */
```

```
    pthread_mutex_lock (&particle)
```

```
    sem_wait (&force)
```

```
    pthread_mutex_unlock (&particle)
```

```
    return NULL
```

```
} /* end minus */
```

```
void main ( ) {
```

```
    pthread_t th_1, th_2, th_3
```

```
    sem_init (&force, 0, 1)
```

```
    pthread_create (&th_1, NULL, plus, NULL)
```

```
    pthread_create (&th_2, NULL, minus, NULL)
```

```
    pthread_join (th_2, NULL)
```

```
    pthread_create (&th_3, NULL, minus, NULL)
```

```
    pthread_join (th_1, &global)                    /* statement D */
```

```
    pthread_join (th_3, 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 – plus	th_2 – minus	th_3 – minus
subito dopo stat. A	<i>ESISTE</i>	<i>PUÒ ESISTERE</i>	<i>PUÒ ESISTERE</i>
subito dopo stat. C in th_2	<i>PUÒ ESISTERE</i>	<i>ESISTE</i>	<i>NON ESISTE</i>
subito dopo stat. D	<i>NON ESISTE</i>	<i>NON ESISTE</i>	<i>PUÒ ESISTERE</i>

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)
- una variabile mutex assume valore 0 per mutex libero e valore 1 per mutex occupato

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>particle</i>	<i>force</i>	<i>global</i>
subito dopo stat. A	<i>0 / 1</i>	<i>0 / 1</i>	<i>0 / 2</i>
subito dopo stat. B	<i>1</i>	<i>1 / 2</i>	<i>0 / 2</i>
subito dopo stat. C in th_3	<i>0 / 1</i>	<i>0 / 1</i>	<i>1 / 2</i>

Il sistema può andare in stallo (*deadlock*), con uno o più *thread* che si bloccano, in **due casi diversi** (con *deadlock* si intende anche un blocco dovuto a un solo *thread* che non potrà mai proseguire). Si indichino gli statement dove avvengono i blocchi e il valore (o i valori) della variabile *global*:

caso	th_1 – plus	th_2 – minus	th_3 – minus
1	<i>wait</i>	<i>terminato</i>	<i>wait</i>
2	<i>lock</i>	<i>terminato</i>	<i>wait</i>

esercizio n. 2 – gestione dei processi

prima parte – stati dei processi

// programma prova.c	
main () {	
pidl = fork ()	
if (pidl == 0) { // codice eseguito dal figlio Q	
execl ("/acso/prog_x", "prog_x", NULL)	
exit (-1)	
} else { // codice eseguito dal padre P	
write (fd, vet, 5) // scrittura di 1 blocco su file precedentemente aperto	
pidl = waitpid (pidl, ...)	
exit (0)	
} // if	
} // prova	

// programma prog_X.c	
int num	
pthread_mutex_t DOOR= PTHREAD_MUTEX_INITIALIZER	
sem_t CHECK	
void * UNO (void * arg) {	void * DUE (void * arg) {
sem_wait (&CHECK)	if (num > 0) {
pthread_mutex_lock (&DOOR)	pthread_mutex_lock (&DOOR)
sem_wait (&CHECK)	sem_post (&CHECK)
pthread_mutex_unlock (&DOOR)	pthread_mutex_unlock (&DOOR)
return NULL	} else {
} // UNO	sem_post (&CHECK) }
	return NULL
	} // DUE
main () { // codice eseguito da Q	
pthread_t TH_1, TH_2	
sem_init (&CHECK, 0, 0)	
// viene letto da standard input il valore di num	
pthread_create (&TH_1, NULL, UNO, NULL)	
pthread_create (&TH_2, NULL, DUE, NULL)	
pthread_join (TH_2, NULL)	
pthread_join (TH_1, NULL)	
exit (1)	
} // main	

Un processo **P** esegue il programma **prova** creando il figlio **Q**, che esegue una mutazione di codice che va a buon fine. Nel codice mutato **prog_X**, **Q** crea i thread **TH_1** e **TH_2**. Si ipotizzi che il valore di **num** letto da **Q** sia maggiore di 0.

Si simuli l'esecuzione dei processi (fino a **udt = 160**) così come risulta dal codice dato, dagli eventi indicati e facendo bene attenzione allo stato iniziale considerato per la simulazione. **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 del tempo indicato**; si noti che la prima riga della tabella **potrebbe essere solo parzialmente completata**

TABELLA DA COMPILARE (numero di colonne non significativo)

<i>identificativo simbolico del processo</i>		IDLE	P	<i>Q</i>	<i>TH_1</i>	<i>TH_2</i>		
<i>evento/processo-chiamata</i>	<i>PID</i>	1	2	3	4	5		
	<i>TGID</i>	1	2	3	3	3		
	0	Pronto	ESEC	NON ESISTE	<i>Non esiste</i>	<i>Non esiste</i>		
<i>P – fork</i>	10	<i>Pronto</i>	<i>ESEC</i>	<i>Pronto</i>	<i>Non esiste</i>	<i>Non esiste</i>		
<i>P – write</i>	20	<i>Pronto</i>	<i>Attesa (write)</i>	<i>ESEC</i>	<i>Non esiste</i>	<i>Non esiste</i>		
<i>Q - execl</i>	30	<i>Pronto</i>	<i>Attesa (write)</i>	<i>ESEC</i>	<i>Non esiste</i>	<i>Non esiste</i>		
interrupt da RT_clock, scadenza quanto di tempo	40	<i>Pronto</i>	<i>Attesa (write)</i>	<i>ESEC</i>	<i>Non esiste</i>	<i>Non esiste</i>		
<i>1 interrupt da DMA_in (operazione write completata)</i>	50	<i>Pronto</i>	<i>ESEC</i>	<i>Pronto</i>	<i>Non esiste</i>	<i>Non esiste</i>		
interrupt da RT_clock, scadenza quanto di tempo	60	<i>Pronto</i>	<i>Pronto</i>	<i>ESEC</i>	<i>Non esiste</i>	<i>Non esiste</i>		
<i>Q – create TH_1</i>	70	<i>Pronto</i>	<i>Pronto</i>	<i>ESEC</i>	<i>Pronto</i>	<i>Non esiste</i>		
interrupt da RT_clock, scadenza quanto di tempo	80	<i>Pronto</i>	<i>ESEC</i>	<i>Pronto</i>	<i>Pronto</i>	<i>Non esiste</i>		
<i>P – waitpid</i>	90	<i>Pronto</i>	<i>Attesa (waitpid)</i>	<i>Pronto</i>	<i>ESEC</i>	<i>Non esiste</i>		
<i>TH_1 – sem_wait</i>	100	<i>Pronto</i>	<i>Attesa (waitpid)</i>	<i>ESEC</i>	<i>Attesa (sem_wait)</i>	<i>Non esiste</i>		
<i>Q – create TH_2</i>	110	<i>Pronto</i>	<i>Attesa (waitpid)</i>	<i>ESEC</i>	<i>Attesa (sem_wait)</i>	<i>Pronto</i>		
<i>Q – join TH_2</i>	120	<i>Pronto</i>	<i>Attesa (waitpid)</i>	<i>Attesa (join)</i>	<i>Attesa (sem_wait)</i>	<i>ESEC</i>		
<i>TH_2 – lock</i>	130	<i>Pronto</i>	<i>Attesa (waitpid)</i>	<i>Attesa (join)</i>	<i>Attesa (sem_wait)</i>	<i>ESEC</i>		
<i>TH_2 – sem_post</i>	140	<i>Pronto</i>	<i>Attesa (waitpid)</i>	<i>Attesa (join)</i>	<i>ESEC</i>	<i>Pronto</i>		
<i>TH_1 – lock</i>	150	<i>Pronto</i>	<i>Attesa (waitpid)</i>	<i>Attesa (join)</i>	<i>Attesa (lock)</i>	<i>ESEC</i>		
<i>TH_2 – unlock</i>	160	<i>Pronto</i>	<i>Attesa (waitpid)</i>	<i>Attesa (join)</i>	<i>ESEC</i>	<i>Pronto</i>		

seconda parte – scheduling dei processi

Si consideri uno Scheduler CFS con **3 task** caratterizzato da queste condizioni iniziali (**da completare**):

CONDIZIONI INIZIALI (da completare)							
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	3	6	4,00	t1	100		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	t1	1	0,25	1,5	1,00	10	100,00
RB	t2	1	0,25	1,5	1,00	30	100,50
	t3	2	0,5	3,0	0,50	20	101,00

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

Events of task t1: EXIT at 1.0;

Events of task t2: WAIT at 0.5; WAKEUP after 2.5;

Events of task t3: CLONE at 2.0

Simulare l'evoluzione del sistema per **4 eventi** riempiendo le seguenti tabelle (per scrivere le eventuali condizioni di preemption, si usi lo spazio tra le tabelle degli eventi):

EVENTO		TIME	TYPE	CONTEXT	RESCHED		
		1.00	EXIT	t1	true		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	2	6	3	t2	100,5		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	t2	1	0.33	2	1	30	100.5
RB	t3	2	0.67	4	0.5	20	101.0
WAITING							

EVENTO		TIME	TYPE	CONTEXT	RESCHED		
		1.50	WAIT	t2	true		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	1	6	2	t3	101.0		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	t3	2	1	6	0.5	20	101.0
RB							
WAITING	t2	1				30.5	101.0

EVENTO		TIME	TYPE	CONTEXT	RESCHED		
		<i>3.5</i>	<i>CLONE</i>	<i>t3</i>	<i>false</i>		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	<i>2</i>	<i>6</i>	<i>4</i>	<i>t3</i>	<i>102</i>		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	<i>t3</i>	<i>2</i>	<i>0.5</i>	<i>3</i>	<i>0.5</i>	<i>22</i>	<i>102</i>
RB	<i>t4</i>	<i>2</i>	<i>0.5</i>	<i>3</i>	<i>0.5</i>	<i>0</i>	<i>103.5</i>
WAITING	<i>t2</i>	<i>1</i>				<i>30.5</i>	<i>101.0</i>

$t_{new}.vrt + WGR * t_{new}.LC = 103,50 + 1,00 * 0,50 = 104,00 < curr.vrt = 102,00$

EVENTO		TIME	TYPE	CONTEXT	RESCHED		
		<i>4.0</i>	<i>W_UP</i>	<i>t3</i>	<i>true</i>		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	<i>3</i>	<i>6</i>	<i>5</i>	<i>t2</i>	<i>102.25</i>		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	<i>t2</i>	<i>1</i>	<i>0,2</i>	<i>1.2</i>	<i>1.0</i>	<i>30.5</i>	<i>101.00</i>
RB	<i>t3</i>	<i>2</i>	<i>0.4</i>	<i>2.4</i>	<i>0.5</i>	<i>22.5</i>	<i>102.25</i>
	<i>t4</i>	<i>2</i>	<i>0.4</i>	<i>2.4</i>	<i>0,5</i>	<i>0</i>	<i>103.5</i>
WAITING							

$t_w.vrt + WGR * t_w.LC = 101,00 + 1,00 * 0,20 = 101,20 < curr.vrt = 102,25$

esercizio n. 3 – gestione della memoria

prima parte – gestione dello spazio virtuale

È dato un sistema di memoria caratterizzato dai seguenti parametri generali: **MAXFREE = 3**, **MINFREE = 2**.
Si consideri la 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 7FFFFFFFC, 3, W, P, A, <-1,0>
PT: <c0 :1 R> <s0 :s0 W> <s1 :- -> <d0 :5 W> <d1 :- -> <p0 :2 W>
    <p1 :3 W> <p2 :- ->
process P - NPV of PC and SP: c0, p0
```

```
_____MEMORIA FISICA_____ (pagine libere: 3)_____
00 : <ZP>                      || 01 : Pc0 / <X,0>                ||
02 : Pp0                       || 03 : Pp1                      ||
04 : ----                     || 05 : Pd0                      ||
06 : ----                     || 07 : ----                      ||
```

```
_____STATO del TLB_____
Pc0 : 01 - 0: 1:              || Pp0 : 02 - 1: 1:              ||
      -----                 || Pd0 : 05 - 1: 0:              ||
Pp1 : 03 - 1: 0:              || -----                 ||
```

SWAP FILE: Ps0, ----, ----, ----, ----, ----

LRU ACTIVE: PP0, PC0

LRU INACTIVE: pp1, pd0

Si rappresenti l'effetto dei seguenti eventi consecutivi sulle strutture dati della memoria compilando esclusivamente le tabelle fornite per ciascun evento (l'assenza di una tabella significa che non è richiesta la compilazione della corrispondente struttura dati).

ATTENZIONE: le Tabelle sono PARZIALI – riempire solamente le celle indicate

evento 1: *fork* (Q)

PT del processo: P				
C0: 1 R	S0: s0 R	D0: 5 R	P0: 4 W	P1: 3 R
PT del processo: Q				
C0: 1 R	S0: s0 R	D0: 5 R	P0: 2 D W	P1: 3 R

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X,0>
02: Qp0 D	03: Pp1 / Qp1
04: Pp0	05: Pd0 / Qd0
06:	07:

TLB							
NPV	NPV	NPV	NPV	NPV	NPV	NPV	NPV
Pc0:	1	0	1	Pp0:	4	1	1

SWAP FILE	
s0: <i>Ps0 / Qs0</i>	s1:
s2:	s3:

Active: *QP0, QC0, PP0, PC0* _____ Inactive: *qp1, qd0, pp1, pd0*

evento 2: *write* (Pd0)

PT del processo: P				
C0: <i>1 R</i>	S0: <i>s0 R</i>	D0: <i>3 W</i>	P0: <i>4 W</i>	P1: <i>s2 R</i>

MEMORIA FISICA	
00: <i><ZP></i>	01: <i>Pc0 / Qc0 / <X,0></i>
02: <i>Qp0 D</i>	03: <i>Pd0</i>
04: <i>Pp0</i>	05:
06:	07:

TLB									
NPV	NPV	NPV	NPV	D	D	D	D	A	A
Pc0:	1	0							
1									
Pp0:	4	1	1						

SWAP FILE	
s0: <i>Ps0 / Qs0</i>	s1: <i>Qd0</i>
s2: <i>Qp1 / Pp1</i>	s3:

Active: *PD0, QP0, QC0, PP0, PC0* _____ Inactive: _____

evento 3: indicare il contenuto delle liste *LRU* dopo ognuna delle seguenti invocazioni consecutive di *kswapd*

a) Read (pc0), 1 kswapd

Active: *PD0, PP0, PC0, qp0, qc0* _____ Inactive: _____

b) Read (pc0), 1 kswapd

Active: *PC0, pd0, pp0* _____ Inactive: *QP0, QC0* _____

c) Read (pc0), 1 kswapd

Active: *PC0* _____ Inactive: *PD0, PP0, qp0, qc0*

d) Read (pc0, pp0), 1 kswapd

Active: PC0, pp0_____

Inactive: pd0, qp0, qc0

seconda parte – gestione del file system

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

MAXFREE = 3 MINFREE = 1

Si consideri la seguente **situazione iniziale**:

_____MEMORIA FISICA_____ (pagine libere: 4)_____			
00 : <ZP>		01 : Pc0 / Qc0 / <X,0>	
02 : Qp0 D		03 : Pp0	
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 sempre in esecuzione il processo **P**.

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); fd1 = open (G)

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F	0	1		
file G	0	1		

evento 3 – write (fd, 11000)

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

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F	11000	1	3	0

evento 4 – *write (fd1, 6000)*

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

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F	11000	1	3	3
file G	6000	1	2	0

eventi 5 e 6 – *lseek (fd, -4000); write (fd, 100)*

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

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F	7100	1	4	3

eventi 7 e 8 – *close (fd); close (fd1)*

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F			4	4
file G			2	2

