



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 mercoledì 22 febbraio 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) _____

esercizio n. 1 – programmazione concorrente

Si consideri il programma C seguente (gli "#include" e le inizializzazioni dei mutex sono omessi):

```
pthread_mutex_t zero
sem_t red, blue
int global = 0
```

```
void * less (void * arg) {
    sem_wait (&blue)
    pthread_mutex_lock (&zero)
    sem_wait (&blue)
```

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

```
    pthread_mutex_unlock (&zero)
    return NULL
```

```
} /* end less */
```

```
void * equal (void * arg) {
    pthread_mutex_lock (&zero)
```

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

```
    pthread_mutex_unlock (&zero)
    return 1
```

```
} /* end equal */
```

```
void * more (void * arg) {
```

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

```
    pthread_mutex_lock (&zero)
    sem_wait (&red)
    pthread_mutex_unlock (&zero)
    return NULL
```

```
} /* end more */
```

```
void main ( ) {
```

```
    pthread_t th_1, th_2, th_3
    sem_init (&red, 0, 0)
    sem_init (&blue, 0, 1)
    pthread_create (&th_3, NULL, more, NULL)
    pthread_create (&th_1, NULL, less, NULL)
    pthread_create (&th_2, NULL, equal, NULL)
```

```
    pthread_join (th_2, &global)                    /* statement D */
```

```
    pthread_join (th_1, NULL)
    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 – less	th_2 – equal	th_3 – more
subito dopo stat. A	Esiste	Può esistere	Esiste
subito dopo stat. B	Esiste	Esiste	Esiste
subito dopo stat. C	Può esistere	Può esistere	Esiste
subito dopo stat. D	Può esistere	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 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>red</i>	<i>blue</i>
subito dopo stat. A	1	0
subito dopo stat. B	0	2 - 1
subito dopo stat. C	1 - 0	2 - 1 - 0

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 – less	th_2 – equal	th_3 – more	<i>global</i>
1	sem_wait(&blue)	mutex_lock(&zero)	mutex_lock(&zero)	2
2	mutex_lock(&zero)	-	sem_wait(&red)	1 - 2

esercizio n. 2 – gestione dei processi

prima parte – stati dei processi

// programma prova.c	
main () {	
pid1 = fork ()	
if (pid1 == 0) { // codice eseguito solo da Q	
fd = open ("/acso/esame", O_RDWR)	
read (fd, vett, 30)	
exit (1)	
} else {	
pid1 = waitpid (pid1, ...)	
fd = open ("/acso/bozza", O_RDWR)	
write (fd, vett, 5)	
exit (0)	
} /* if */	
} /* prova */	

// programma prog_x.c	
pthread_mutex_t GATE = PTHREAD_MUTEX_INITIALIZER	
sem_t CHECK	
void * SINGLE (void * arg) {	void * SEQUENCE (void * arg) {
(1) sem_wait (&CHECK)	(5) pthread_mutex_lock (&GATE)
(2) pthread_mutex_lock (&GATE)	(6) sem_post (&CHECK)
(3) sem_wait (&CHECK)	(7) pthread_mutex_unlock (&GATE)
(4) pthread_mutex_unlock (&GATE)	(8) sem_post (&CHECK)
return NULL	return NULL
} /* SINGLE */	} /* SEQUENCE */

main () { // codice eseguito da S	
pthread_t TH_1, TH_2	
sem_init (&CHECK, 0, 0)	
pthread_create (&TH_1, NULL, SINGLE, (void *) 1)	
pthread_create (&TH_2, NULL, SEQUENCE, NULL)	
exit (1)	
} /* main */	

Un processo **P** esegue il programma **prova**. Un processo **S** esegue il programma **prog_x**. Il processo **P** crea il processo **Q**. Il processo **S** crea i thread **TH1** e **TH2**.

Si simuli l'esecuzione dei processi (fino a **udt = 90**) così come risulta dal codice dato, dagli eventi indicati e ipotizzando che il processo **P** non abbia ancora eseguito la **waitpid**. **Si completi** la tabella riportando quanto segue:

- $\langle PID, TGID \rangle$ di ciascun processo che viene creato
- $\langle \text{identificativo del processo-chiamata di sistema} / \text{libreria} \rangle$ 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**

Nota bene: nella riga con **udt = 40** lo stato raggiunto dai vari processi è già indicato e si deve individuare l'evento che li porta in tale stato.

TABELLA DA COMPILARE (numero di colonne non significativo)

<i>identificativo simbolico del processo</i>		IDLE	P	S	Q	TH1	TH2		
<i>evento processo-chiamata</i>	<i>PID</i>	1	2	3	4	5	6		
	<i>TGID</i>	1	2	3	4	3	3		
S –sem_init	0	pronto	pronto	esec	A read	NE	NE		
S - pthread_create(TH1)	10	pronto	pronto	ESEC	A read	pronto	NE		
<i>interrupt da RT_clock e scadenza quanto di tempo</i>	20	pronto	ESEC	pronto	A read	pronto	NE		
<i>interrupt da DMA_in, tutti i blocchi richiesti trasferiti</i>	30	pronto	pronto	pronto	ESEC	pronto	NE		
Interrupt da RT_CLOCK e scadenza quanto di tempo	40	pronto	pronto	pronto	pronto	esec	NE		
TH1 - sem_wait(&CHECK)	50	pronto	pronto	ESEC	pronto	A sem	NE		
S - pthread_create(TH2)	60	pronto	pronto	ESEC	pronto	A sem	pronto		
S - exit	70	pronto	ESEC	NE	pronto	NE	NE		
P - wait	80	pronto	A wait	NE	ESEC	NE	NE		
Q - exit	90	pronto	ESEC	NE	NE	NE	NE		

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	t1	100		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	t1	1	0.25	1.5	1	10	100,00
RB	t2	2	0.5	3	0.5	30	100,50
	t3	1	0.25	1.5	1	20	101,00

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

Events of task t1: WAIT at 1.0; WAKEUP after 6.0;

Events of task t2: WAIT at 0.5; WAKEUP after 1.0;

Simulare l'evoluzione del sistema per **5 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	WAIT	T1	TRUE		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	2	6	3	T2	100.5		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T2	2	2/3	4	0.5	30	100.5
RB	T3	1	1/3	2	1	20	101
WAITING	T1	1				11	101

$$T1 \rightarrow VRT = 100 + 1 * 1 = 101$$

EVENTO		TIME	TYPE	CONTEXT	RESCHED		
		1.5	WAIT	T2	TRUE		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	1	6	1	T3	100.75		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T3	1	1	6	1	20	101
RB							
WAITING	T1	1				11	101
	T2	2				30.5	100.75

$$T2 \rightarrow VRT = 100.5 + 0.5 * 0.5 = 100.75$$

EVENTO		TIME	TYPE	CONTEXT	RESCHED		
		2.5	WAKE UP	T3	TRUE		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	2	6	3	T2	102		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T2	2	2/3	4	0.5	30.5	100.75
RB	T3	1	1/3	2	1	21	101.5
WAITING	T1	1				11	101

T3 -> VRT = 101 + 1 * 1 = 102

T2 -> VRT = 100.75

WAKE UP => $100.75 + 1 * 0.66 = 101.44 < 101.5 \Rightarrow \text{TRUE}$

EVENTO		TIME	TYPE	CONTEXT	RESCHED		
		6.5	S.Q.D.T	T2	TRUE		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	2	6	3	T3	102		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T3	1	1/3	2	1	21	101.5
RB	T2	2	2/3	4	0.5	34.5	102.75
WAITING	T1	1				11	101

T2 -> VRT = 100.75 + 4 * 0.5 = 102.75

WAKE UP => $101 + 1 * 0.25 = 101.25 < 102 \Rightarrow \text{TRUE}$

EVENTO		TIME	TYPE	CONTEXT	RESCHED		
		7	WAKE UP	T3	TRUE		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	3	6	4	T1	102.5		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T1	1	0.25	1.5	1	11	101
RB	T3	1	0.25	1.5	1	21.5	102
	T2	2	0.5	3	0.5	34.5	102.75
WAITING							

T3 -> VRT = 101.5 + 0.5 * 1 = 102

T1 -> VRT = 101

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 *****
PT:  <c0 :1  R>   <s0 :s1 R>   <s1 :- ->   <d0 :s2 R>   <d1 :- ->
      <p0 :2  R>   <p1 :6  W>   <p2 :3  W>   <p3 :- ->
process P - NPV of PC and SP:  c0, p2
```

```
PROCESSO: Q *****
PT:  <c0 :1  R>   <s0 :s1 R>   <s1 :- ->   <d0 :s2 R>   <d1 :- ->
      <p0 :2  R>   <p1 :s0 W>   <p2 :- ->
process Q - NPV of PC and SP:  c0, p1
```

MEMORIA FISICA (pagine libere: 3)			
00 : <ZP>		01 : Pc0 / Qc0 / <X,0>	
02 : Pp0 / Qp0		03 : Pp2	
04 : ----		05 : ----	
06 : Pp1		07 : ----	

STATO del TLB			
Pc0 : 01 - 0: 1:		Pp0 : 02 - 1: 0:	
Pp2 : 03 - 1: 1:		-----	
Pp1 : 06 - 1: 0:		-----	

SWAP FILE: Qp1, Ps0 / Qs0, Pd0 / Qd0, ----, ----, ----

LRU ACTIVE: PC0

LRU INACTIVE: pp2, pp1, pp0, qp0, qc0

ATTENZIONE: lo swap file NON è vuoto.

Si rappresenti l'effetto dei seguenti eventi 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).

evento 1: *read* (Ps0, Pd0)

PT del processo: P				
c0 :1 R	p0 :s3 R	p1 :s4 W	p2 :3 W	p3 :- -
s0 :4 R	s1 :- -	d0 :2 R	d1 :- -	
PT del processo: Q				
c0 :1 R	p0 :s3 R	p1 :s0 W	p2 :- -	s0 :4 R
s1 :- -	d0 :2 R	d1 :- -		

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: Pd0 / Qd0	03: Pp2
04: Ps0 / Qs0	05: ----
06: ----	07:

TLB							
NPV	NPF	D	A	NPV	NPF	D	A
Pc0	1	0	1	Pp2	3	1	1
Pd0	4	0	1	Ps0	5	0	1

SWAP FILE	
s0: Qp1	s1: Ps0 / Qs0
s2: Pd0 / Qd0	s3: Pp0 / Qp0
s4: Pp1	s5:

LRU ACTIVE: PD0, PS0, PC0

LRU INACTIVE: pp2, qc0, qs0, qd0

evento 2: *write* (Ps0, Pp1)

PT del processo: P				
c0 :1 R	s0 :5 W	s1 :- -	d0 :2 R	d1 :- -
p0 :s3 R	p1 :3 W	p2 :s5 W	p3 :- -	
PT del processo: Q				
c0 :1 R	s0 :s1 W	s1 :- -	d0 :2 R	d1 :- -
p0 :s3 R	p1 :s0 W	p2 :- -		

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: Pd0 / Qd0	03: Pp1
04: ----	05: Ps0
06:	07:

TLB							
NPV	NPF	D	A	NPV	NPF	D	A
Pc0	1	0	1	Pd0	2	0	1
Pp1	3	1	1	Ps0	5	1	1

SWAP FILE	
s0: Qp1	s1: Qs0
s2: Pd0 / Qd0	s3: Pp0 / Qp0
s4: ----	s5: Pp2

LRU ACTIVE: PP1, PD0, PS0, PC0

LRU INACTIVE: qc0, qd0

seconda parte – gestione del 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: 3)			
00 : <ZP>	01 : Pc0 / Qc0 / <X,0>		
02 : Pp0 / Qp0	03 : Qp1 D		
04 : Pp1	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 è cumulativo, quindi è la somma delle pagine lette o scritte da tutti gli eventi precedenti oltre a quello considerato.

evento 1 – fd = *open* (F)

f_pos	f_count	numero pagine lette	numero pagine scritte
0	1		

evento 2 – *read* (fd, 3500)

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

f_pos	f_count	numero pagine lette	numero pagine scritte
3500	1	1	0

evento 3 – write (fd, 4500)

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: Pp0 / Qp0	03: Qp1 (D)
04: Pp1	05: <F, 0> (D)
06: <F, 1> (D)	07:

f_pos	f_count	numero pagine lette	numero pagine scritte
8000	1	2	0

evento 4 – write (fd, 4500)

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: Pp0 / Qp0	03: Qp1 (D)
04: Pp1	05: <F, 2> (D)
06: <F, 3> (D)	07:

f_pos	f_count	numero pagine lette	numero pagine scritte
12500	1	4	2

evento 5 – close (fd)

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: Pp0 / Qp0	03: Qp1 (D)
04: Pp1	05: <F, 2>
06: <F, 3>	07:

f_pos	f_count	numero pagine lette	numero pagine scritte
		4	4

