



Politecnico di Milano

Dipartimento di Elettronica, Informazione e Bioingegneria

prof.ssa Anna Antola
prof. Luca Breveglieri

prof.ssa Donatella Sciuto
prof.ssa Cristina Silvano

AXO – Architettura dei Calcolatori e Sistemi Operativi

SECONDA PARTE – giovedì 10 settembre 2020

Cognome _____ Nome _____

Matricola _____ Codice Persona _____

Istruzioni – ESAME ONLINE

È vietato consultare libri, eserciziari e appunti, nonché cellulari e altri dispositivi mobili di calcolo o comunicazione. Chiunque non dovesse attenersi alla regola vedrà annullata la propria prova.

La prova va sempre consegnata completando la procedura prevista nel modulo (form) dell'esame con INVIO (SUBMIT) del testo risolto. Se lo studente intende RITIRARSI deve inviare messaggio di posta elettronica (email) al docente dopo avere completata la procedura.

Dallo HONOR CODE

In qualsiasi progetto o compito, gli studenti devono dichiarare onestamente il proprio contributo e devono indicare chiaramente le parti svolte da altri studenti o prese da fonti esterne.

Ogni studente garantisce che eseguirà di persona tutte le attività associate all'esame senza alcun aiuto di altri; la sostituzione di identità è un reato perseguibile per legge.

Durante un esame, gli studenti non possono accedere a fonti (libri, note, risorse online, ecc) diverse da quelle esplicitamente consentite.

Durante un esame, gli studenti non possono comunicare con nessun altro, né chiedere suggerimenti.

In caso di esame a distanza, gli studenti non cercano di violare le regole a causa del controllo limitato che il docente può esercitare.

L'accettazione dello Honor Code costituisce prerequisito per l'iscrizione agli esami.

Tempo a disposizione 1 h : 30 m

Valore indicativo di domande ed esercizi, voti parziali e voto finale:

esercizio 1 (?? punti) _____

esercizio 2 (?? punti) _____

esercizio 3 (?? punti) _____

voto finale: (16 punti) _____

CON SOLUZIONI (in corsivo)

BLANK PAGE

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 primitive di libreria NPTL):

```
pthread_mutex_t neutral
sem_t one, zero
int global = 0
```

```
void * min (void * arg) {
    mutex_lock (&neutral)
    sem_wait (&one)
```

```
    global = 1                                     /* statement A */
```

```
    sem_wait (&one)
```

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

```
    mutex_unlock (&neutral)
    return NULL
```

```
} /* end min */
```

```
void * max (void * arg) {
    mutex_lock (&neutral)
    sem_post (&one)
```

```
    mutex_unlock (&neutral)                       /* statement C */
```

```
    global = 2
```

```
    mutex_lock (&neutral)
```

```
        sem_wait (&zero)
```

```
    mutex_unlock (&neutral)
```

```
    global = 3                                     /* statement D */
```

```
    return NULL
```

```
} /* end max */
```

```
void main ( ) {
    pthread_t th_1, th_2
    sem_init (&one, 0, 1)
    sem_init (&zero, 0, 0)
    create (&th_1, NULL, min, NULL)
    create (&th_2, NULL, max, NULL)
```

```
    join (th_1, NULL)                             /* statement E */
```

```
    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 – min	th_2 – max
subito dopo stat. A	<i>ESISTE</i>	<i>PUÒ ESISTERE</i>
subito dopo stat. B	<i>ESISTE</i>	<i>ESISTE</i>
subito dopo stat. C	<i>PUÒ ESISTERE</i>	<i>ESISTE</i>
subito dopo stat. E	<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)
- 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>neutral</i>	<i>one</i>	<i>zero</i>
subito dopo stat. A	<i>1</i>	<i>0 / 1</i>	<i>0</i>
subito dopo stat. C	<i>0 / 1</i>	<i>0 / 1 / 2</i>	<i>0 / 1</i>
subito dopo stat. D	<i>0</i>	<i>0</i>	<i>0</i>
subito dopo stat. E	<i>0 / 1</i>	<i>0</i>	<i>0 / 1</i>

Il sistema può andare in stallo (deadlock), con uno o più *thread* che si bloccano, in (almeno) **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 i corrispondenti valori di *global*:

caso	th_1 – min	th_2 – max	<i>global</i>
1	<i>2a wait one</i>	<i>1a lock neutral</i>	<i>1</i>
2	<i>lock neutral</i>	<i>wait zero</i>	<i>2</i>
3			

esercizio n. 2 – processi e nucleo

prima parte – gestione dei processi

// programma prova.c	
main () {	
pid1 = fork ()	// P crea Q
if (pid1 == 0) {	// codice eseguito da Q
pid2 = fork ()	// Q crea R
if (pid2 == 0) {	// codice eseguito da R
read (stdin, msg, 50)	
exit (2)	
} else {	// codice eseguito da Q
pid = wait (&status)	// Q aspetta la terminazione di R
} // end_if pid2	
} else {	// codice eseguito da P
execl ("/acso/prog_x", "prog_x", NULL)	
exit (-1)	
} // end_if pid1	
exit (0)	// codice eseguito da Q
} // prova	

// programma prog_x.c	
// dichiarazione e inizializzazione dei mutex presenti nel codice	
void * more (void * arg) {	void * less (void * arg) {
mutex_lock (&positive)	mutex_lock (&positive)
sem_wait (&one)	sem_wait (&one)
mutex_unlock (&positive)	sem_wait (&one)
sem_post (&one)	mutex_unlock (&positive)
sem_wait (&one)	sem_post (&one)
return NULL	return NULL
} // more	} // less
main () { // codice eseguito da P	
pthread_t th_1, th_2	
sem_init (&one, 0, 2)	
create (&th_1, NULL, more, NULL)	
create (&th_2, NULL, less, NULL)	
join (th_2, NULL)	
join (th_1, NULL)	
exit (1)	
} // main	

Un processo **P** esegue il programma **prova**. Il processo **P** crea il processo figlio **Q** e poi il processo **P** esegue i thread **th_1** e **th_2**. Il processo **Q** crea un processo figlio **R**.

Si simuli l'esecuzione dei processi completando tutte le righe presenti nella tabella così come risulta dal codice dato, **dallo stato iniziale** e dagli eventi indicati, e tenendo conto che

- il processo **P** ha già eseguito la primitiva **create (&th_1, ...)** ma non la primitiva **create (&th_2, ...)**
- il processo (thread) **th_1** ha già eseguito la primitiva **unlock (&positive)** ma non la primitiva **sem_post (&one)**

Si completi la tabella riportando quanto segue:

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

<i>identificativo simbolico del processo</i>		IDLE	P	Q	th_1	R	th_2	
<i>evento oppure processo-chiamata</i>	<i>PID</i>	1	2	3	4	5	6	
	<i>TGID</i>	1	2	3	2	5	2	
Q – pid2 = fork (Q crea R)	0	pronto	pronto (da più tempo)	esec	pronto	<i>pronto</i>	<i>NE</i>	
interrupt da <i>RT_clock</i> e scadenza quanto di tempo per il processo in esecuzione	1	<i>pronto</i>	<i>esec</i>	<i>pronto</i>	<i>pronto</i>	<i>pronto</i>	<i>NE</i>	
<i>P – create th_2</i>	2	<i>pronto</i>	<i>esec</i>	<i>pronto</i>	<i>pronto</i>	<i>pronto</i>	<i>pronto</i>	
<i>P – join th_2</i>	3	<i>pronto</i>	<i>A join th2</i>	<i>pronto</i>	<i>esec</i>	<i>pronto</i>	<i>pronto</i>	
th_1 – sem_post (one) dopo l'invocazione il semaforo <i>one</i> vale 2	4	<i>pronto</i>	<i>A join th2</i>	<i>pronto</i>	<i>esec</i>	<i>pronto</i>	<i>pronto</i>	
interrupt da <i>RT_clock</i> e scadenza quanto di tempo per il processo in esecuzione	5	<i>pronto</i>	<i>A join th2</i>	<i>pronto</i>	<i>pronto</i>	<i>esec</i>	<i>pronto</i>	
<i>R – read (stdin)</i>	6	<i>pronto</i>	<i>A join th2</i>	<i>esec</i>	<i>pronto</i>	<i>A read</i>	<i>pronto</i>	
<i>Q – wait</i>	7	<i>pronto</i>	<i>A join th2</i>	<i>A wait R exit</i>	<i>pronto</i>	<i>A read</i>	<i>esec</i>	
<i>th_2 – mutex_lock (positive)</i>	8	<i>pronto</i>	<i>A join th2</i>	<i>A wait</i>	<i>pronto</i>	<i>A read</i>	<i>esec</i>	
<i>th_2 – sem_wait (one)</i> <i>one = 1</i>	9	<i>pronto</i>	<i>A join th2</i>	<i>A wait</i>	<i>pronto</i>	<i>A read</i>	<i>esec</i>	
<i>th_2 – sem_wait (one)</i> <i>one = 0</i>	10	<i>pronto</i>	<i>A join th2</i>	<i>A wait</i>	<i>pronto</i>	<i>A read</i>	<i>esec</i>	
<i>50 interrupt da std_in, tutti i caratteri trasferiti</i>	11	pronto	<i>A join th2</i>	<i>A wait</i>	pronto	esec	pronto	
<i>R – exit (2)</i>	12	<i>pronto</i>	<i>A join th2</i>	<i>esec</i>	<i>pronto</i>	<i>NE</i>	<i>pronto</i>	
<i>Q – exit (0)</i>	13	<i>pronto</i>	<i>A join th2</i>	<i>NE</i>	<i>esec</i>	<i>NE</i>	<i>pronto</i>	
<i>th_1 – sem_wait (one)</i> <i>one = 0</i>	14	<i>pronto</i>	<i>A join th2</i>	<i>NE</i>	<i>A sem_wait</i>	<i>NE</i>	<i>esec</i>	

seconda parte – scheduling CFS

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	1.50	1	50	101.0
RB	t2	1	0.25	1.50	1	60	102.0
	t3	2	0.5	3.00	0.50	70	103.0

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

Events of task t1: EXIT at 2.0;
Events of task t2: WAIT at 1.0; WAKEUP after 1.0;
Events of task t3: WAIT at 1.5; WAKEUP after 0.5;

Simulare l'evoluzione del sistema per **4 eventi** riempiendo le seguenti tabelle.

Indicare la valutazione delle condizioni di preemption per l'evento di WAKEUP nell'apposito spazio alla fine dell'esercizio.

EVENTO		TIME	TYPE	CONTEXT	RESCHED		
		1.5	Q_SCADE	t1	true		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	3	6	4	t2	102		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	t2	1	0.25	1.50	1	60	102.0
RB	t1	1	0.25	1.50	1	51.5	102.5
	t3	2	0.5	3.00	0.50	70	103.0
WAITING							

EVENTO		TIME	TYPE	CONTEXT	RESCHED		
		2.5	WAIT	t2	true		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	2	6	3	t1	102.5		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	t1	1	0.33	2.0	1	51.5	102.5
RB	t3	2	0.67	4.00	0.50	70	103.0
WAITING	t2	1				61.0	103

EVENTO		TIME	TYPE	CONTEXT	RESCHED		
		3	EXIT	t1	true		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	1	6	2	t3	103		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	t3	2	1	6.00	0.50	70	103.0
RB							
WAITING	t2	1				61.0	103

EVENTO		TIME	TYPE	CONTEXT	RESCHED		
		3.5	W_UP	t3	false		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	2	6	3	t3	103.25		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	t3	2	0.67	4.00	0.50	70.50	103.25
RB	t2	1	0.33	2.00	1	61.0	103
WAITING							

Condizioni di rescheduling a **wake_up** del **task t2**:

wake_up: $103,00 + 1,00 * 0,33 = 103,33 < \text{curr.vrt} = 103,25? \rightarrow \text{false}$

Condizioni di rescheduling a **wake_up** del **task t3**:

wake_up: *non verificata negli eventi considerati*

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 = 2 MINFREE = 1

Situazione iniziale (esistono due processi P e Q)

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>
M0 000001000, 4, W, P, M, <F, 0>
M1 000002000, 4, W, S, M, <G, 0>
P 7FFFFFFFC, 3, W, P, A, <-1, 0>

PT: <c0 :1 R> <s0 :- -> <s1 :- -> <d0 :- -> <d1 :- -> <p0 :2 R>
<p1 :4 W> <p2 :- -> <m00:6 W> <m01:- -> <m02:- -> <m03:- ->
<m10:3 W> <m11:- -> <m12:- -> <m13:- ->

process P - NPV of PC and SP: c0, p1

PROCESSO: Q ****(i dettagli di questo processo non interessano) *****

MEMORIA FISICA (pagine libere: 2)

00 : <ZP>	01 : Pc0 / Qc0 / <X,0>
02 : Pp0 / Qp0	03 : Pm10 / <G,0>
04 : Pp1	05 : ----
06 : Pm00	07 : ----

STATO del TLB

Pc0 : 01 - 0: 1:	Pp0 : 02 - 1: 0:
Pp1 : 04 - 1: 0:	Pm00 : 06 - 1: 0:
Pm10 : 03 - 1: 0:	-----
-----	-----

SWAP FILE: Qp1, ----, ----, ----, ----, ----, ----,

LRU ACTIVE: PC0,

LRU INACTIVE: pp1, pm10, pm00, pp0, qp0, qc0,

evento 1: write (Pp0)

(Pp0 causa COW e viene riallocata)

PT del processo: P				
P0: 5 W	P1: 4 W	P2: - -	M00: 6 W	M10: 3 W

process P - NPV of PC and SP: c0, p0

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: Qp0 D	03: Pm10 / <G, 0>
04: Pp1	05: Pp0
06: Pm00	07:

SWAP FILE	
s0: Qp1	s1:
s2:	s3:

evento 2: write (Ps1)

(viene invocato PFRA che libera le pagine fisiche 02 e 05, poi si carica ps1, poi la scrittura causa COW)

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: <X, 2>	03: Pm10 / <G, 0>
04: Pp1	05: Ps1
06: Pm00	07:

SWAP FILE	
s0: Qp1	s1: Qp0
s2: Pp0	s3:

LRU ACTIVE: PS1, PC0,

LRU INACTIVE: pp1, pm10, pm00, qc0

evento 3: write (Pp2, Pp3)

(la prima write causa PFRA che libera le pagine 02 e 06)

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: pp2	03: Pm10 / <G, 0>
04: Pp1	05: Ps1
06: Pp3	07:

SWAP FILE	
s0: Qp1	s1: Qp0
s2: Pp0	s3: Pm00
s4:	s5:

evento 4: write (Pp4)

(causa PFRA che libera le pagine 03 e 04. Pm10 è shared, quindi copiata in G,0)

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: Pp2	03: Pp4
04:	05: Ps1
06: Pp3	07:

SWAP FILE	
s0: Qp1	s1: Qp0
s2: Pp0	s3: Pm00
s4: Pp1	s5:

LRU ACTIVE: PP4, PP3, PP2, PS1, PC0,

LRU INACTIVE: $qc0$

process P - NPV of PC and SP: $c0, p4$

seconda parte – 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: 5)			
00 : <ZP>		01 : Pc2 / <X, 2>	
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**.

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. Si ricorda che *close* scrive le pagine dirty di un file solo se fcount diventa = 0.

Eventi 1 e 2: *fd = open ("F"), write (fd, 6000)*

MEMORIA FISICA	
00: <ZP>	01: Pc2 / <X, 2>
02: Pp0	03: <F, 0> D
04: <F, 1> D	05: ----
06: ----	07: ----

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

Evento 3: *close (fd)*

MEMORIA FISICA	
00: <ZP>	01: Pc2 / <X, 2>
02: Pp0	03: <F, 0>
04: <F, 1>	05: ----
06: ----	07: ----

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F	--	0	2	2

Eventi 4 e 5: *fork ("Q"), context_switch ("Q")*

NOTA BENE: al momento del *context_switch*, la pagina p0 di P è marcata dirty nel TLB

MEMORIA FISICA	
00: <ZP>	01: Pc2 / Qc2 / <X, 2>
02: Qp0 D	03: <F, 0>
04: <F, 1>	05: Pp0 D
06: ----	07: ----

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F	--	0	2	2

Eventi 6 e 7: *fd = open ("F"), read (8000)*

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F	8000	1	2	2

Evento 8: *write (fd, 10000)*

Vengono scritte le pagine del file: F1, F2, F3 e F4. La pagina F1, già presente in memoria, viene scritta in memoria; la pagina F2 viene caricata in 06 e scritta in memoria. Il caricamento di F3 fa attivare PFRA e devono essere scaricate le tre pagine F0, F1 (scrittura su disco) e F2 (scrittura su disco). La pagina F3 viene caricata in 03 e scritta in memoria; la pagina F4 viene caricata in 04 e scritta in memoria. Con questo evento ci sono 3 letture da disco e 2 scritture su disco.

MEMORIA FISICA	
00: <ZP>	01: Pc2 / Qc2 / <X, 2>
02: Qp0 D	03: <F, 3> D
04: <F, 4> D	05: Pp0 D
06: ----	07: ----

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F	18000	1	2+3=5	2+2=4

Evento 9: *close (fd)*

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F	--	0	5	6