



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

Prova di lunedì 31 gennaio 2022

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 **2 h : 00 m**

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

esercizio 1 (4 punti) _____

esercizio 2 (5 punti) _____

esercizio 3 (5 punti) _____

esercizio 4 (2 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 sense
sem_t see, hear
int global = 0
```

```
void * hand (void * arg) {
    mutex_lock (&sense)
    sem_post (&see)
    global = 1
    mutex_unlock (&sense)
    global = 2
    sem_wait (&hear)
    mutex_lock (&sense)
    sem_wait (&hear)
    mutex_unlock (&sense)
    return NULL
} /* end hand */
```

```
void * foot (void * arg) {
    mutex_lock (&sense)
    sem_wait (&see)
```

```
    global = 3                                     /* statement B */
```

```
    mutex_unlock (&sense)
    sem_wait (&hear)
```

```
    global = 4                                     /* statement C */
```

```
    mutex_lock (&sense)
    sem_post (&hear)
    mutex_unlock (&sense)
    return (void *) 5
```

```
} /* end foot */
```

```
void main ( ) {
    pthread_t th_1, th_2
    sem_init (&see, 0, 0)
    sem_init (&hear, 0, 2)
    create (&th_1, NULL, hand, NULL)
    create (&th_2, NULL, foot, NULL)
```

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

```
    join (th_2, &global)
    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 – <i>hand</i>	th_2 – <i>foot</i>
subito dopo stat. A	Esiste	Può esistere
subito dopo stat. B	Può esistere	Esiste
subito dopo stat. C	Esiste	Esiste
subito dopo stat. D	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 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>sense</i>	<i>see</i>	<i>hear</i>	<i>global</i>
subito dopo stat. A	1	1	2	1
subito dopo stat. B	1	0	2 - 1 - 0	2 - 3
subito dopo stat. C	0 - 1	0	1 - 0	4 - 2
subito dopo stat. D	1 - 0	1 - 0	0	4 - 2 - 3

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 **due casi**, indicando gli statement dove avvengono i blocchi e i possibili valori della variabile *global*:

caso	th_1 – <i>hand</i>	th_2 – <i>foot</i>	<i>global</i>
1	<code>mutex_lock(&sense)</code>	<code>sem_wait(&see)</code>	0
2	-	<code>sem_wait(&hear)</code>	2 - 3
3	<code>sem_wait(&hear)</code>	<code>mutex_lock(&sense)</code>	2 - 4

esercizio n. 2 – processi e nucleo

prima parte – gestione dei processi

// programma main.c	
int main () { // processo P	
pid_t pid = fork ()	
if (pid == 0) { // codice eseguito solo da Q	
execl ("/acso/nuovo", "nuovo", NULL)	
exit (-1)	
} // end if	
write (stdout, "Fin!", 4)	
wait (&status)	
exit (0)	
} // end main.c	
// programma nuovo.c	
sem_t pass	
mutex_t lock	
char vet [2] = {1 ,2}	
void * one (void * arg) {	void * two (void * arg) {
sem_wait (&pass)	mutex_lock (&lock)
write (stdout, &vet[0], 1)	read (stdin, &vet[1], 1)
sem_post (&pass)	mutex_unlock (&lock)
return NULL	sem_post (&pass)
} // end one	return NULL
	} // end two
int main () { // codice eseguito da Q	
pthread_t TH_1, TH_2	
sem_init (&pass, 0, 1)	
pthread_create (&TH_2, NULL, two, NULL)	
pthread_create (&TH_1, NULL, one, NULL)	
pthread_join (TH_2, NULL)	
pthread_join (TH_1, NULL)	
exit (1)	
} // fine nuovo.c	

Un processo **P** esegue il programma **main.c** e crea un processo figlio **Q** che esegue una mutazione di codice (programma **nuovo.c**). La mutazione di codice va a buon fine e viene creata una coppia di 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 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

identificativo simbolico del processo		Idle	P	Q	TH_1	TH_2
evento oppure processo-chiamata	PID	1	2	3	5	4
	TGID	1	2	3	3	3
P – pid = fork ()	0	pronto	esec	pronto	NE	NE
P – write	1	pronto	A write	ESEC	NE	NE
Q – execl	2	pronto	A write	ESEC	NE	NE
Q – pthread_create TH_2	3	pronto	A write	ESEC	NE	pronto
interrupt da RT_clock e scadenza quanto di tempo	4	pronto	A write	pronto	NE	ESEC
TH_2 – mutex_lock	5	pronto	A write	pronto	NE	ESEC
TH_2 – read	6	pronto	A write	ESEC	NE	A read
Q - pthread_create(TH1)	7	pronto	attesa (write)	esec	pronto	attesa (read)
Q – pthread_join TH_2	8	pronto	A write	A join	ESEC	A read
TH_1 – sem_wait	9	pronto	A write	A join	ESEC	A read
TH_1 – write	10	ESEC	A write	A join	A write	A read
Interrupt da STDIN, tutti i blocchi letti	11	pronto	A write	A join	A write	ESEC
TH_2 – mutex_unlock	12	pronto	A write	A join	A write	ESEC
TH_2 – sem_post	13	pronto	A write	A join	A write	ESEC
TH_2 – return	14	pronto	A write	ESEC	A write	NE
Interrupt STDOUT, tutti i blocchi scritti	15	pronto	A write	pronto	ESEC	NE
TH_1 – sem_post	16	pronto	A write	pronto	ESEC	NE
TH_1 – return	17	pronto	A write	ESEC	NE	NE
Q - join(TH1)	18	pronto	A write	ESEC	NE	NE
Q – exit	19	ESEC	A write	NE	NE	NE

seconda parte – scheduling dei processi

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		
	2	6	3	T1	100		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T1	1	0,33	2	1	10	100
RB	T2	2	0,67	4	0,5	20	101

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

Events of task t1: WAIT at 1.0 WAKEUP after 3.5

Events of task t2: CLONE 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	T1 -> VRT = 100 + 1 * 1 = 101	
		1	WAIT	T1	TRUE		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	1	6	2	T2	101		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T2	2	1	6	0.5	20	101
RB							
WAITING	T1	1				11	101

EVENTO 2		TIME	TYPE	CONTEXT	RESCHED	T2 -> VRT = 101 + 2 * 0.5 = 102 T2F -> VRT = 102 + 3 * 0.5 = 103.5	
		3	CLONE	T2	FALSE		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	2	6	4	T2	102		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T2	2	0.5	3	0.5	22	102
RB	T2F	2	0.5	3	0.5	0	103.5
WAITING	T1	1				11	101

$$T2 \rightarrow VRT = 102 + 1 * 0.5 = 102.5$$

EVENTO 3 (*)		TIME	TYPE	CONTEXT	RESCHED	(*) E' stata considerata corretta anche la soluzione che prevede che T2 CURR non rientri subito nella lista RB e pertanto T3 diventa CURR	
		4	S.Q.D.T	T2	TRUE		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	2	6	4	T2	102.5		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T2	2	0.5	3	0.5	23	102.5
RB	T2F	2	0.5	3	0.5	0	103.5
WAITING	T1	1				11	101

EVENTO 4		TIME	TYPE	CONTEXT	RESCHED	T2 -> VRT = 102.5 + 0.5 * 0.5 = 103 T1 -> VRT = 101	
		4.5	WAKE UP	T2	TRUE		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	3	6	5	T1	103		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T1	1	0.2	1.2	1	11	101
RB	T2	2	0.4	2.4	0.5	24	102.75
	T2F	2	0.4	2.4	0.5	0	103.5
WAITING							

Calcolo del VRT iniziale del **task T3** creato dalla **CLONE**

$$T2F \rightarrow VRT = 102 + 3 * 0.5 = 103.5$$

Valutazione della cond. di rescheduling alla **CLONE**

$$103.5 + 1 * 0.5 = 104 > 102 \Rightarrow \text{FALSE}$$

Valutazione della cond. di rescheduling alla **WAKEUP**

$$101 + 1 * 0.5 = 101.2 < 102.75 \Rightarrow \text{TRUE}$$

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 = 3

MINFREE = 2

situazione iniziale (esistono un processo P e un processo Q)

PROCESSO: P *****

VMA : C 000000400, 2, R, P, M, <XX, 0>
K 000000600, 1, R, P, M, <XX, 2>
S 000000601, 1, W, P, M, <XX, 3>
P 7FFFFFFFC, 3, W, P, A, <-1, 0>

PT: <c0 :- -> <c1 :1 R> <k0 :- -> <s0 :- -> <p0 :2 D R>
<p1 :4 D W> <p2 :- ->

process P - NPV of PC and SP: c1, p1

PROCESSO: Q ****SOLO LE INFORMAZIONI RILEVANTI PER L'ESERCIZIO ****

VMA : C 000000400, 2, R, P, M, <XX, 0>
K 000000600, 1, R, P, M, <XX, 2>
S 000000601, 1, W, P, M, <XX, 3>
M0 000010000, 4, W, S, M, <G, 0>
P 7FFFFFFFC, 3, W, P, A, <-1, 0>

process Q - NPV of PC and SP: c1, p1

MEMORIA FISICA (pagine libere: 3)

00 : <ZP>		01 : Pc1 / Qc1 / <XX, 1>	
02 : Pp0 / Qp0 D		03 : Qp1 D	
04 : Pp1 D		05 : Qm00 / <G, 0> D	
06 : Qm01 / <G, 1> D		07 : ----	
08 : ----		09 : ----	

STATO del TLB

Pc1 : 01 - 0: 1:		Pp1 : 04 - 1: 1:	
-----		-----	
-----		-----	
-----		-----	

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

LRU ACTIVE: QM01, QM00, QC1,

LRU INACTIVE: qp1, qp0, pp1, pp0, pc1,

evento 1: read (Pc1) – write (Pp2)

PT del processo: P				
p0: :2 R D	p1: :4 W D	p2: :7 W	<p3 :- ->	

process P	NPV of PC: c1	NPV of SP: p2
------------------	---------------	---------------

MEMORIA FISICA	
00: <ZP>	01: Pc1 / Qc1 / <XX, 1>
02: Pp0 / Qp0 (D)	03: Qp1 (D)
04: Pp1 (D)	05: Qm00 / <G, 0> (D)
06: Qm01 / <G, 1> (D)	07: Pp2
08:	09:

LRU ACTIVE: PP2, QM01, QM00, QC1

LRU INACTIVE: qp1, qp0, pp1, pp0, pc1

evento 2: read (Pc1, Pp0) – write (Pp3)

PT del processo: P				
p0: :s1 R	p1: :s0 W	p2: :7 W	p3: :2 W	p4 :- -

process P	NPV of PC: c1	NPV of SP: p3
------------------	---------------	---------------

MEMORIA FISICA	
00: <ZP>	01: Pc1 / Qc1 / <XX, 1>
02: Pp3	03: Qp1 (D)
04: ----	05: Qm00 / <G, 0> (D)
06: Qm01 / <G, 1> (D)	07: Pp2
08:	09:

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

LRU ACTIVE: PP3, PP2, QM01, QM00, QC1

LRU INACTIVE: qp1, pc1

evento 3: clone (S, c0)

VMA del processo P/S (è da compilare solo la riga relativa alla VMA T0)							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
T0	7FFF F77F E	2	W	P	A	-1	0

PT dei processi: P/S				
t00: :4 W	t01: :- -			

process P	NPV of PC : c1	NPV of SP : p3
process S	NPV of PC : c0	NPV of SP : t00

TLB							
NPV	NPF	D	A	NPV	NPF	D	A
PSc1	1	0	1	PSp3	2	1	1
PSt00	4	1	1	PSp2	7	1	1

LRU ACTIVE: PST00, PSP3, PSP2, QM01, QM00, QC1

LRU INACTIVE: qp1, psc1

spazio libero per appunti correzioni o continuazioni (se necessario)

seconda parte – file system

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

MAXFREE = 2 **MINFREE = 1**

Si consideri la seguente **situazione iniziale**.

process P – NPV of PC and SP: c2, p0

MEMORIA FISICA (pagine libere: 1)			
00 : <ZP>		01 : Pc2 / <X, 2>	
02 : Pp0		03 : <G, 2>	
04 : Pm00		05 : <F, 0> D	
06 : <F, 1> D		07 : ----	

STATO del TLB			
Pc2 : 01 - 0: 1:		Pp0 : 02 - 1: 1:	
Pm00 : 04 - 1: 1:		-----	
-----		-----	

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P	F	6000	1	2	0

ATTENZIONE: è presente la colonna “processo” dove va specificato il nome/i del/i processo/i a cui si riferiscono le informazioni “f_pos” e “f_count” (campi di struct file) relative al file indicato.

Il processo **P** è in esecuzione. Il file **F** è stato aperto da **P** tramite chiamata **fd1 = open (F)**.

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

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

eventi 1 e 2: *fork (Q)* *context switch (Q)*

MEMORIA FISICA	
00: <ZP>	01: Pc2 / Qc2 / <X, 2>
02: Qp0 (D)	03: Pp0 (D)
04: Pm00 / Qm00 (D)	05: ----
06: <F, 1> (D)	07:

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P, Q	F	6000	2	2	1

evento 3: *read (fd1, 500)*

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P, Q	F	6500	2	2	1

eventi 4, 5 e 6: *fd2 = open (F)* *read (fd2, 9000)* *close (fd1)*

MEMORIA FISICA	
00: <ZP>	01: Pc2 / Qc2 / <X, 2>
02: Qp0 (D)	03: Pp0 (D)
04: Pm00 / Qm00 (D)	05: <F, 2>
06:	07:

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P	F	6500	1	4	2
Q	F	9000	1		

esercizio n. 4 – domande varie

prima domanda – moduli del SO

stato iniziale: **CURR = T, Q = PRONTO**

La runqueue contiene due task: il thread **T** e il processo normale **Q**; il sistema non contiene altri task.

Si consideri il seguente evento: il **thread T** è in esecuzione in **modo U** ed esegue l'istruzione di **return** della sua funzione "**fun**" per terminare la sua esecuzione.

Domanda:

- Mostrare le **invocazioni di tutti i moduli** (ed eventuali relativi **ritorni**) eseguiti nel contesto del thread **T** per gestire l'evento indicato.
- Mostrare (in modo simbolico) il l'eveluzione dello **stack di sistema** del thread **T** al termine della gestione dell'evento considerato.

invocazione moduli – numero di righe non significativo

processo	modo	modulo
T	U	fun< // return di fun
T	U	clone // si torna a clone
T	U	> syscall
T	U -> S	SYSCALL (> system_call)
T	S	> sys_exit
T	S	> wake_up_process
T	S	> check_preempt_curr <
T	S	wake_up_process <
T	S	> schedule
T	S	> deque_task <
T	S	> pick_next_task <
T -> Q	S	CONTEXT_SWITCH (schedule)

evoluzione sStack_T al termine dell'evento – numero di righe non significativo

USP
l. rientro a sys_exit da schedule
l. rientro a system_call da sys_exit
PSR (U)
l. rientro a syscall da system_call

seconda domanda – tabella delle pagine

Date le VMA di un processo P sotto riportate, definire:

1. la decomposizione degli indirizzi virtuali dell'NPV iniziale di ogni area secondo la notazione **PGD : PUD : PMD : PT**
2. il numero di pagine necessarie in ogni livello della gerarchia e il numero totale di pagine necessarie a rappresentare la Tabella delle Pagine (TP) del processo
3. il numero di pagine virtuali occupate dal processo
4. il rapporto tra l'occupazione della TP e la dimensione virtuale del processo in pagine
5. la dimensione virtuale massima del processo in pagine, senza dover modificare la dimensione della TP
6. il rapporto relativo

VMA del processo P							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
C	0000 0040 0	3	R	P	M	X	0
K	0000 0060 0	1	R	P	M	X	3
S	0000 0060 1	4	W	P	M	X	4
D	0000 0060 5	2	W	P	A	-1	0
M0	0000 1000 0	2	W	S	M	G	2
M1	0000 3000 0	1	W	P	M	F	4
M2	0000 4000 0	1	W	P	A	-1	0
T1	7FFF F77F B	2	W	P	A	-1	0
T0	7FFF F77F E	2	W	P	A	-1	0
P	7FFF FFFF C	3	W	P	A	-1	0

1. Decomposizione degli indirizzi virtuali.

		PGD :	PUD :	PMD :	PT
C	0000 0040 0	0	0	2	0
K	0000 0060 0	0	0	3	0
S	0000 0060 1	0	0	3	1
D	0000 0060 5	0	0	3	5
M0	0000 1000 0	0	0	128	0
M1	0000 3000 0	0	0	384	0
M2	0000 4000 0	0	1	0	0
T1	7FFF F77F B	255	511	443	507
T0	7FFF F77F E	255	511	443	510
P	7FFF FFFF C	255	511	511	508

2. Numero di pagine necessarie:

pag PGD: 1

pag PUD: 2

pag PMD: 3

pag PT: 7

pag totali: 13

3. Numero di pagine virtuali occupate dal processo: 21

4. Rapporto di occupazione: 61.9%

5. Dimensione massima del processo in pagine virtuali: $7 * 512 = 3584$

6. Rapporto di occupazione con dimensione massima: 0.36%