



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 24 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) _____

esercizio n. 1 – programmazione concorrente

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

```
pthread_mutex_t row
sem_t point, line
int global = 0
```

```
void * circle (void * arg) {
    pthread_mutex_lock (&row)
```

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

```
    pthread_mutex_unlock (&row)
    pthread_mutex_lock (&row)
```

```
    sem_wait (&line)                                /* statement B */
```

```
    pthread_mutex_unlock (&row)
    return 1
```

```
} /* end circle */
```

```
void * square (void * arg) {
```

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

```
    pthread_mutex_lock (&row)
    sem_wait (&point)
    sem_post (&line)
    pthread_mutex_unlock (&row)
    return NULL
```

```
} /* end square */
```

```
void main ( ) {
```

```
    pthread_t th_1, th_2
    sem_init (&point, 0, 0)
    sem_init (&line, 0, 0)
    pthread_create (&th_1, NULL, circle, NULL)
    pthread_create (&th_2, NULL, square, NULL)
```

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

```
    pthread_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 – circle	th_2 – square
subito dopo stat. A		
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)
- 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>row</i>	<i>point</i>	<i>global</i>
subito dopo stat. A			
subito dopo stat. B			
subito dopo stat. C			

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:

caso	th_1 – circle	th_2 – square
1		
2		

esercizio n. 2 – gestione dei processi

prima parte – stati dei processi

// programma prog_X.c		
pthread_mutex_t GATE = PTHREAD_MUTEX_INITIALIZER		
sem_t GO		
void * A (void * arg) {		void * B (void * arg) {
(1) pthread_mutex_lock (&GATE)		(4) pthread_mutex_lock (&GATE)
(2) sem_wait (&GO)		(5) sem_post (&GO)
(3) pthread_mutex_unlock (&GATE)		(6) pthread_mutex_unlock (&GATE)
nanosleep (1)		(7) sem_wait (&GO)
return NULL		return NULL
} // thread A		} // thread B
main () { // codice eseguito da P		
pthread_t TH_A, TH_B		
sem_init (&GO, 0, 1)		
pthread_create (&TH_B, NULL, B, NULL)		
pthread_create (&TH_A, NULL, A, NULL)		
write (stdout, vett, 1)		
(8) pthread_join (&TH_A, NULL)		
(9) sem_post (&GO)		
(10) pthread_join (&TH_B, NULL)		
exit (1)		
} // main		

// programma prog_Y.c		
pthread_mutex_t DOOR= PTHREAD_MUTEX_INITIALIZER		
sem_t CHECK		
void * UNO (void * arg) {		void * DUE (void * arg) {
(11) sem_wait (&CHECK)		if (num > 3) {
(12) pthread_mutex_lock (&DOOR)		(15) sem_post (&CHECK) }
(13) sem_wait (&CHECK)		} else {
(14) pthread_mutex_unlock (&DOOR)		(16) pthread_mutex_lock (&DOOR)
return NULL		(17) sem_post (&CHECK)
} // UNO		(18) pthread_mutex_unlock (&DOOR) }
		return NULL
		} // DUE
main () { // codice eseguito da S		
pthread_t TH_1, TH_2		
sem_init (&CHECK, 0, 1)		
pthread_create (&TH_1, NULL, UNO, (void *) 1)		
pthread_create (&TH_2, NULL, DUE, NULL)		
(19) pthread_join (TH_2, NULL)		
(20) pthread_join (TH_1, NULL)		
exit (1)		
} // main		

Un processo **P** esegue il programma **prog_X** creando i thread **TH_A** e **TH_B**. Un processo **S** esegue il programma **prog_Y** creando i thread **TH_1** e **TH_2**.

Si simuli l'esecuzione dei processi (fino a **udt = 100**) così come risulta dal codice dato, dagli eventi indicati e facendo bene attenzione allo stato iniziale considerato per la simulazione. Oltre a quanto indicato nella prima riga della tabella, per lo stato iniziale di simulazione valgono le seguenti ipotesi:

- il thread **TH_B** è in esecuzione, ha già eseguito la **sem_post (&GO)** ma non ha ancora eseguito la **pthread_mutex_unlock (&GATE)**
- il thread **TH_2** è in stato di pronto, ha già eseguito la **sem_post (&CHECK)** (n° d'ordine 17) ma non ha ancora eseguito la **pthread_mutex_unlock (&DOOR)**

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

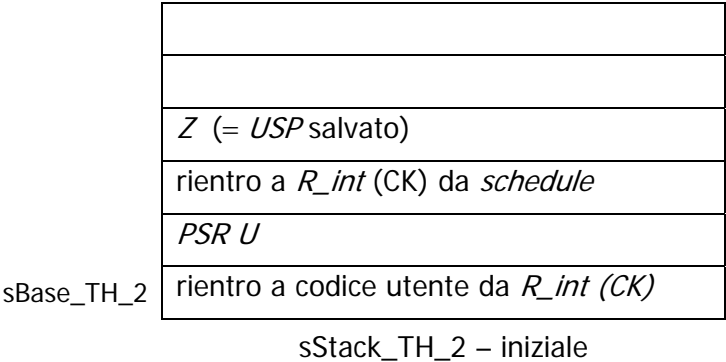
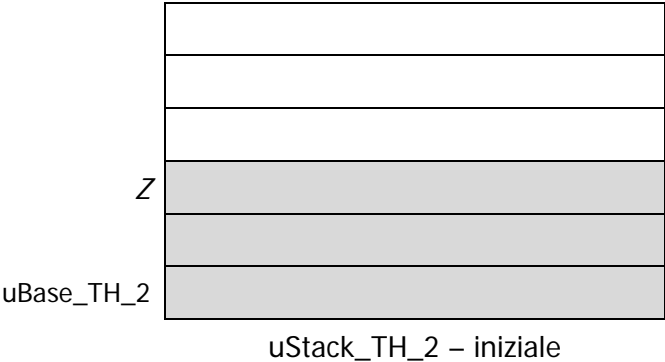
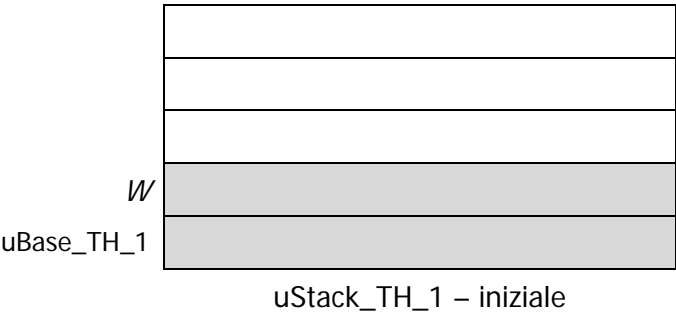
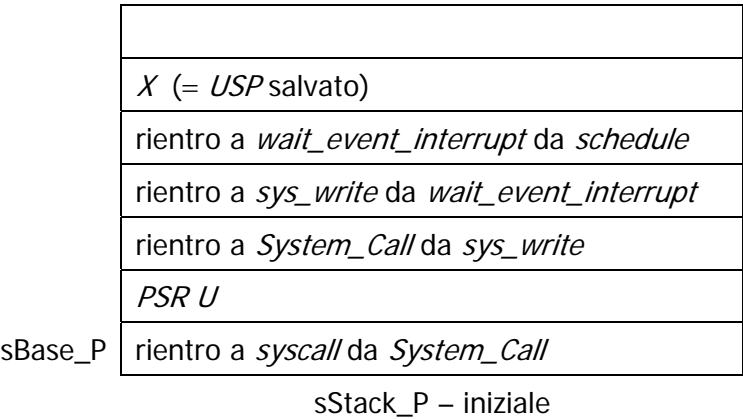
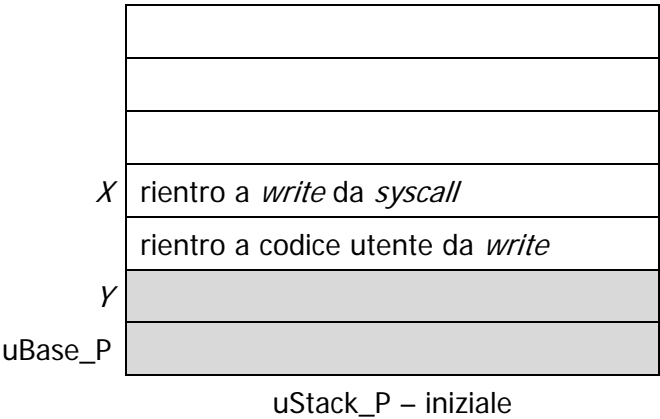
<i>identificativo simbolico del processo</i>		IDLE	P	S	TH_B	TH_A	TH_1	TH_2
<i>evento/processo-chiamata</i>	<i>PID</i>	1	2	3	2	2	3	3
	<i>TGID</i>	1	2	3	4	5	6	7
	0	pronto	attesa (write)	attesa (join TH_2)	ESEC v. ipotesi stato iniziale	attesa (lock gate)	attesa (lock door)	pronto v. ipotesi stato iniziale
Interrupt da RT_clock, scadenza quanto di tempo	10							
	20							
	30							
	40	pronto	ESEC	attesa (join TH_2)	pronto	attesa (lock gate)	pronto	pronto
	50							
	60							
	70							
Interrupt da RT_clock, scadenza quanto di tempo	80							
	90							
	100							

Domanda – Si consideri la simulazione effettuata e le chiamate di sistema riportate nella Tabella sopra, e numerate nel codice del programma. Con riferimento alla loro implementazione tramite *futex*, si indichino i numeri d'ordine di quelle eseguite:

- senza invocare *System_Call*:
- con invocazione di *System_Call*:

seconda parte – struttura e moduli del nucleo

Si considerino i tre processi *P*, *TH_1* e *TH_2* della prima parte. Lo stato iniziale delle pile di sistema e utente dei tre processi è riportato qui sotto.



domanda 1 - Si indichi lo stato dei processi così come deducibile dallo stato iniziale delle pile specificando anche l'evento o la chiamata di sistema che ha portato il processo in tale stato:

P

TH_1

TH_2

domanda 2 – A partire dallo stato iniziale descritto, si consideri l'evento sotto specificato. **Si mostrino** le invocazioni di tutti i **moduli** (e eventuali relativi ritorni) per la gestione dell'evento stesso (precisando processo e modo) e il **contenuto delle pile** utente e di sistema richieste.

NOTAZIONE da usare per i moduli: > (invocazione), nome_modulo (esecuzione), < (ritorno)

EVENTO: *interrupt* da *standard_output* e completamento di *write* (a seguito dell'evento il processo **P** ha maggiori diritti di esecuzione di tutti gli altri in *runqueue*).

Si mostri lo stato delle pile di **TH_1** al termine della gestione dell'evento.

invocazione moduli (num. di righe vuote non signif.)

<i>processo</i>	<i>modo</i>	<i>modulo</i>

contenuto della pila

uBase_TH_1

uStack_TH_1

sBase_TH_1

sStack_TH_1

domanda 3 – A seguito dell'evento le **pile** di **P** e di **TH_2** si sono modificate? Come risultano rispetto allo stato iniziale?

P:

TH_2:

esercizio n. 3 – gestione della memoria

prima parte – gestione dello spazio virtuale

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

MAXFREE = 1

MINFREE = 1

Si consideri la seguente **situazione iniziale**:

PROCESSO: P

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

PT: <c0 :- -> <c1 :1 R> <p0 :2 W> <p1 :- -> <p2 :- ->

process P - NPV of PC and SP: c1, p0

MEMORIA FISICA (pagine libere: 5)			
00 : <ZP>		01 : Pc1 / <XX,1>	
02 : Pp0		03 : ----	
04 : ----		05 : ----	
06 : ----		07 : ----	

STATO del TLB			
Pc1 : 01 - 0: 1:		Pp0 : 02 - 1: 1:	
-----		-----	

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: sono state create tre nuove VMA (M0, M1 e M2):

1. mmap (0x10000000, 1, W, S, M, "G", 2)
2. mmap (0x30000000, 3, W, P, M, "F", 2)
3. mmap (0x40000000, 2, W, P, A, -1, 0)

VMA del processo P (compilare solo le righe relative alle nuove VMA create)							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
M0							
M1							
M2							

Evento 2: Read (pm20, pm21, pm11) Write (pm00, pm10)

PT del processo: P (completare con pagine di VMA)				
C0: - -	C1: 1 R	P0: 2 W	P1: - -	P2: - -

MEMORIA FISICA	
00:	01:
02:	03:
04:	05:
06:	07:

Evento 3: write (pm11)

MEMORIA FISICA	
00:	01:
02:	03:
04:	05:
06:	07:

Indicare la decomposizione dell'indirizzo della prima pagina della VMA M2 nella TP:

PGD	PUD	PMD	PT

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: 3)____
    00 : <ZP>                ||    01 : Pc0 / Qc0 / <X,0>    ||
    02 : Qp0 D                ||    03 : Pp0                ||
    04 : Pp1                  ||    05 : ----                ||
    06 : ----                  ||    07 : ----                ||

LRU ACTIVE:      PP1
LRU INACTIVE:    pp0, pc0, qp0, qc0
```

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				
file G				

evento 3 – $read(fd, 8000)$

MEMORIA FISICA	
00:	01:
02:	03:
04:	05:
06:	07:

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

evento 4 – *write* (fd1, 4000)

MEMORIA FISICA	
00:	01:
02:	03:
04:	05:
06:	07:

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

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

MEMORIA FISICA	
00:	01:
02:	03:
04:	05:
06:	07:

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F				

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

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

