



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 martedì 29 agosto 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 stop
sem_t min, max
int global = 0
```

```
void * put (void * arg) {
    pthread_mutex_lock (&stop)
```

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

```
    sem_wait (&min)
    pthread_mutex_unlock (&stop)
    sem_wait (&min)
```

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

```
    return NULL
```

```
} /* end put */
```

```
void * get (void * arg) {
```

```
    if (arg == 2) global = 2                       /* statement C */
```

```
    pthread_mutex_lock (&stop)
    sem_post (&min)
    sem_wait (&max)
    pthread_mutex_unlock (&stop)
    return arg
```

```
} /* end get */
```

```
void main ( ) {
```

```
    pthread_t th_1, th_2, th_3
    sem_init (&min, 0, 0)
    sem_init (&max, 0, 1)
    pthread_create (&th_1, NULL, put, NULL)
    pthread_create (&th_2, NULL, get, 2)
    pthread_create (&th_3, NULL, get, 3)
    pthread_join (th_1, NULL)
```

```
    pthread_join (th_2, NULL)                       /* statement D */
```

```
    pthread_join (th_3, &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 – put	th_2 – get	th_3 – get
subito dopo stat. A	Esiste	Può esistere	Può esistere
subito dopo stat. C in th_3	Esiste	Può esistere	Esiste
subito dopo stat. D	Non esiste	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)
- 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>min</i>	<i>max</i>	<i>global</i>
subito dopo stat. A	0 / 1	1 / 0	1 / 2 / 3
subito dopo stat. B	0	0 / 1 / 2	2 / 3 / 1
subito dopo stat. C in th_3	0 / 1	1 / 0	0 / 1 / 2

Il sistema può andare in stallo (*deadlock*), con uno o più *thread* che si bloccano, in **tre 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 – put	th_2 – get	th_3 – get
1 <code>pthread_mutex_lock(&stop)</code>		Terminato	<code>sem_wait(&max)</code>
2 <code>sem_wait(&min)</code>		<code>pthread_mutex_lock(&stop)</code>	<code>pthread_mutex_lock(&stop)</code>
3 <code>pthread_mutex_lock(&stop)</code>		<code>sem_wait(&max)</code>	Terminato

esercizio n. 2 – gestione dei processi

prima parte – stati dei processi

// programma PROG_X.c
main () {
i = 1;
while (i <= 2) {
pid = fork ();
if (pid == 0) { // codice eseguito da F1 e F2
if (i == 1) { // caso i == 1
execl ("/acso/mu_1_2", "mu_1_2", "1", NULL);
write (stdout, err_msg1, 20);
exit (-1);
} else { // caso i == 2
execl ("/acso/mu_1_2", "mu_1_2", "2", NULL);
write (stdout, err_msg2, 20);
exit (-2);
} // fine if 1_2
} // fine if e fine codice eseguito da F1 e F2
i ++;
} // fine while
pid = wait (&status);
exit (0);
} // prog_X

// programma MU_1_2.c
main () { // codice eseguito da F1 e F2
local = valore parametro passato con execl: 1 o 2;
read (stdin, vett, 5);
if (local == 1) { // caso local == 1
pid = fork ();
if (pid == 0) { // codice eseguito da R
exit (3);
} // fine codice eseguito da R
} else { // caso local == 2
write (stdout, msg, 30);
} // fine if 1_2
exit (2);
} // MU_1_2

Un processo **P** esegue il programma **PROG_X** e tramite un ciclo crea due processi figli **F1** e **F2**, che eseguono mutazione di codice (che vanno a buon fine) con uno stesso programma **MU_1_2** cui passano un parametro con valore diverso. Nel codice mutato, il processo **F1** crea un processo figlio **R**.

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. **Si completi** la tabella riportando quanto segue:

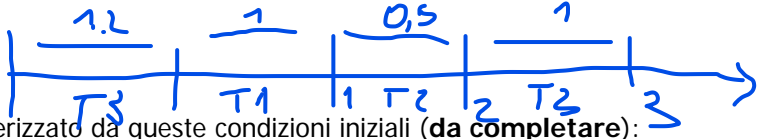
- $\langle PID, TGID \rangle$ di ciascun processo che viene creato
- $\langle \text{identificativo del processo-chiamata di sistema / libreria} \rangle$ nella prima colonna, dove necessario e in funzione del codice proposto
- in ciascuna riga lo stato dei processi **al termine del tempo indicato**; si 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	F1	F2	R		
<i>evento/processo-chiamata</i>	<i>PID</i>	1	2	3	4	5		
	<i>TGID</i>	1	2	3	4	5		
	0	pronto	ESEC	NON ESISTE	NE	NE		
P - fork	10	pronto	ESEC	pronto	NE	NE		
interrupt da RT_clock, scadenza quanto di tempo	20	pronto	pronto	ESEC	NE	NE		
F1 - execl	30	pronto	pronto	ESEC	NE	NE		
F1 - read	40	pronto	ESEC	A read	NE	NE		
P - fork	50	pronto	ESEC	A read	pronto	NE		
P - wait	60	pronto	A wait	A read	pronto	NE		
F2 - execl	70	ESEC	A wait	A read	A write	NE		
5 interrupt da standard input, tutti i dati sono stati letti	80	pronto	A wait	ESEC	A write	NE		
F1 - fork	90	pronto	A wait	ESEC	A write	pronto		
F1 - exit	100	pronto	ESEC	NE	A write	pronto		

seconda parte – scheduling dei processi

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



CONDIZIONI INIZIALI (da completare)							
RUNQUEUEUE	NRT	PER	RQL	CURR	VMIN		
	3	6	5	t3	100		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	t3	1	0.2	1.2	1	20	100,00
RB	t1	2	0.4	2.4	0.5	10	100,50
	t2	2	0.4	2.4	0.5	30	101,00

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

Events of task t1: WAIT at 1.0; WAKEUP after 4.0;

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

Simulare l'evoluzione del sistema per **4 eventi** riempiendo le seguenti tabelle – si usi la Tabella finale per calcolare le condizioni di Rescheduling per gli Eventi di Wakeup

EVENTO		TIME	TYPE	CONTEXT	RESCHED		
		1.2	s QdT	T3	TRUE		
RUNQUEUEUE	NRT	PER	RQL	CURR	VMIN		
	3	6	5	T1	100.5		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T1	2	0.4	2.4	0.5	10	100.5
RB	T2	2	0.4	2.4	0.5	30	101
	T3	1	0.2	1.2	1	21.2	101.2
WAITING							

$$T3 \rightarrow VRT = 100 + 1.2 * 1 = 101.2$$

EVENTO		TIME	TYPE	CONTEXT	RESCHED		
		2.2	wait	T1	TRUE		
RUNQUEUEUE	NRT	PER	RQL	CURR	VMIN		
	2	6	3	T2	101		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T2	2	2/3	4	0.5	30	101
RB	T3	1	1/3	2	1	21.2	101.2
WAITING	T1	2				11	101

$$T1 \rightarrow VRT = 100.5 + 1 * 0.5 = 101$$

EVENTO		TIME	TYPE	CONTEXT	RESCHED	$T2 \rightarrow VRT = 101 + 0.5 * 0.5 = 101.25$	
		2.7	wait	T2	TRUE		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	1	6	1	T3	101.2		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T3	1	1	6	1	21.2	101.2
RB							
WAITING	T1	2				11	101
	T2	2				30.5	101.25

EVENTO		TIME	TYPE	CONTEXT	RESCHED	$T3 \rightarrow VRT = 101.2 + 1 * 1 = 102.2$ $T2 \rightarrow VRT = 101.5$	
		3.7	wake up	T3	TRUE		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	2	6	3	T2	102.2		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T2	2	2/3	4	0.5	30.5	101.25
RB	T3	1	1/3	2	1	22.2	102.2
WAITING	T1	2				11	101

TABELLA per calcolo condizioni di Resched

TIME (dell'evento)	tw.vrt	tw.LC	tw.vrt + WGR*tw.LC	CURR.vrt	TRUE/FALSE

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 = 1**, **dimensione memoria disponibile = 6 pagine**.

Nel sistema è attivo solamente un processo P.

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: alcune Tabelle sono PARZIALI – riempire in tal caso solamente le celle indicate

Evento 1: il processo P esegue exec (3, 1, 4, 2, c2, "X")

(si consultino gli eventi exec e mmap sul fascicolo per il significato dei parametri della exec e delle colonne nella tabella delle VMA)

VMA del processo P							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
C	0x 0000 0040 0000	3	R	P	M	X	0
K	0x 0000 0060 0000	1	R	P	M	X	3
S	0x 0000 0060 0001	4	W	P	M	X	4
D	0x 0000 0060 0005	2	W	P	A	-1	0
P	0x 7FFF FFFF FFFF	3	W	P	A	-1	0

PT del processo: P (parziale: compilare solo le pagine indicate)				
c0: <:- ->	c2: <:1 R>	s0: <:- ->	p0: <:2 W>	p2: <:- ->

NPV del PC: c2

NPV dello SP: p0

MEMORIA FISICA	
00: <ZP>	01: Pc2 / <X, 2>
02: Pp0	03:
04:	05:

TLB							
NPV	NPF	D	A	NPV	NPF	D	A
Pc2	01	0	1	Pp0	02	1	1

Evento 2: clone (Q, c0)

(si ricorda che:

- il primo parametro è il nome del processo che viene creato e il secondo parametro è la pagina della thread function
- la pila di un thread è di 2 pagine (nel nostro modello)

VMA del processo P/Q (inserire solo le VMA nuove o modificate)							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
T0	0x 7FFF F77F FFFF	2	W	P	A	-1	0

PT del processo: P (inserire solo pagine create o modificate dall'evento clone)				

NPV del PC del processo Q: c0 NPV dello SP del processo Q: t00

MEMORIA FISICA	
00: <ZP>	01: PQc2 / <X, 2>
02: PQp0	03: PQt00
04:	05:

TLB							
NPV	NPF	D	A	NPV	NPF	D	A
PQp0	02	1	1	PQc2	01	0	1
PQt00	03	1	1				

seconda parte – memoria e 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: 5)			
00 : <ZP>		01 : Pc0 / <X, 0>	
02 : Pp0		03 : ----	
04 : ----		05 : ----	
06 : ----		07 : ----	
STATO del TLB			
Pc0 : 01 - 0: 1:		Pp0 : 02 - 1: 1:	
-----		-----	

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**.
0 ---- 4096 ---- 8192 ---- 12288 ---- 16384 ---- 20480
0 1 2 3 4

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"), write(fd, 4000)

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

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F	4000	1	1	0

Eventi 3 e 4: fork ("Q"), context switch ("Q")

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

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F	4000	2	1	0

Evento 5: write (fd, 3000) 4000 + 3000 = 7000

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

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F	7000	2	2	0

Evento 6: close (fd)

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F	7000	1	2	0

Evento 7: context switch ("P")

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F	7000	1	2	0

Evento 8: read (fd, 10)

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F	7010	1	2	0

Evento 9: close (fd)

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

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

