



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 2 luglio 2018

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 (**5** punti) \_\_\_\_\_

esercizio 3 (**5.5** punti) \_\_\_\_\_

esercizio 4 (**1.5** 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 primitive di libreria NPTL):

```
pthread_mutex_t open, close
sem_t pass
int global = 0
```

---

```
void * start (void * arg) {
    sem_wait (&pass)
    mutex_lock (&close)
    sem_wait (&pass)
```

```
    mutex_unlock (&close)                                /* statement A */
```

```
    global = 2
    mutex_lock (&open)
    global = 3
    mutex_unlock (&open)
```

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

```
    return arg
```

```
} /* end start */
```

---

```
void * quit (void * arg) {
    mutex_lock (&open)
    sem_post (&pass)
    mutex_lock (&close)
    global = 4
    sem_post (&pass)
```

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

```
    mutex_unlock (&open)
    sem_wait (&pass)
    return NULL
```

```
} /* end quit */
```

---

```
void main ( ) {
    pthread_t th_1, th_2
    sem_init (&pass, 0, 0)
    create (&th_1, NULL, start, (void * 1)
    create (&th_2, NULL, quit, NULL)
```

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

```
    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 – start	th_2 – quit
subito dopo stat. <b>A</b>	Esiste	Esiste
subito dopo stat. <b>B</b>	Esiste	Può esistere
subito dopo stat. <b>C</b>	Esiste	Esiste
subito dopo stat. <b>D</b>	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>pass</i>	<i>global</i>
subito dopo stat. <b>A</b>	0	4
subito dopo stat. <b>C</b>	2 - 1 - 0	4 - 2
subito dopo stat. <b>D</b>	1 - 0	1

**Il sistema può andare in stallo (*deadlock*)**, con uno o più *thread* che si bloccano (con *deadlock* si intende anche un blocco dovuto a un solo *thread* che non potrà mai proseguire), in **due casi**. Si indichino gli **statement dove avvengono i blocchi**:

**QUANTI CASI DI DEADLOCK** (indicare il numero) ? 2

caso	th_1 – start	th_2 – quit	<i>global</i>
1	sem_wait(&pass) (1°)	-	4
2	sem_wait(&pass) (2°)	mutex_lock(&close)	0

## esercizio n. 2 – processi e nucleo

### prima parte – gestione dei processi

// programma <b>double_buffer.c</b>	
char buffer1[BUFFER_SIZE]	
Char buffer2[BUFFER_SIZE]	
fd input_file_fd	
sem_t buffer1_ready, buffer1_empty	
sem_t buffer2_ready, buffer2_empty	
void * READER (void * arg) {	void * EXECUTER (void * arg) {
sem_wait(&buffer1_empty)	sem_wait(&buffer1_ready)
read(input_file_fd, &buffer1, 300)	pid1 = fork()
sem_post(&buffer1_ready)	if (pid1 == 0) { // eseguito da <b>Q</b>
sem_wait(&buffer2_empty)	execl("/acso/exec", "exec", buffer1)
read(input_file_fd, &buffer2, 300)	write(stdout, error_msg, 50)
sem_post(&buffer2_ready)	} else {
return NULL	sem_wait(&buffer2_ready)
}	do_work(buffer2)
/* READER */	sem_post(&buffer2_empty)
	pid1 = wait( &status )
	sem_post(&buffer1_empty)
	}
	return NULL
	}
	/* CONSUMER */
main ( ) { // codice eseguito da <b>P</b>	
pthread_t TH_1, TH_2	
sem_init(&buffer1_ready, 0)	
sem_init(&buffer1_empty, 1)	
sem_init(&buffer2_ready, 0)	
sem_init(&buffer2_empty, 1)	
input_file_fd = open( "in.dat", O_RDONLY )	
pthread_create( &TH_1, NULL, EXECUTER, NULL )	
pthread_create( &TH_2, NULL, READER, NULL )	
pthread_join( TH_1, NULL )	
pthread_join( TH_2, NULL )	
exit (0)	
}	
/* main */	
// programma <b>execution.c</b>	
main ( ) { // codice eseguito da <b>Q</b>	
elaborate_input_file( argv[1]);	
exit (0)	
}	

Un processo **P** esegue il programma **double\_buffer** e crea i thread **TH\_1** e **TH\_2**. Il thread **TH\_1** crea il figlio **Q**, che esegue una mutazione di codice (programma **execution**) che va a buon fine

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. Si completi la tabella riportando quanto segue:

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

<i>identificativo simbolico del processo</i>		<i>IDLE</i>	<i>P</i>	<i>TH_1</i>	<i>TH_2</i>	<i>Q</i>
<i>evento oppure processo-chiamata</i>	<i>PID</i>	1	2	3	4	5
	<i>TGID</i>	1	2	2	2	2
P –open	0	esec	A(open)	NE	NE	NE
<i>interrupt da DMA_in, tutti i blocchi richiesti trasferiti</i>	10	pronto	ESEC	NE	NE	NE
P - pthread_create(TH1)	20	pronto	ESEC	pronto	NE	NE
P - pthread_create(TH2)	30	pronto	ESEC	pronto	pronto	NE
P - pthread_join(TH1)	40	pronto	A join	ESEC	pronto	NE
TH1 - sem_wait(&buffer1_ready)	50	pronto	A join	A wait	ESEC	NE
TH2 - sem_wait(&buffer1_empty)	60	pronto	A join	A wait	ESEC	NE
TH2 - read	70	ESEC	A join	A wait	A read	NE
Interrupt da DMA_IN, tutti i blocchi sono pronti	80	pronto	A(TH1)	A(sem)	esec	NE
TH2 - sem_post(&buffer1_ready)	90	pronto	A join	ESEC	pronto	NE
TH1 - fork	100	pronto	A join	ESEC	pronto	pronto
TH1 - sem_wait(&buffer2_ready)	110	pronto	A join	A wait	ESEC	pronto
TH2 - sem_wait(&buffer2_empty)	120	pronto	A join	A wait	ESEC	pronto
TH2 - read	130	pronto	A join	A wait	A read	ESEC

## seconda parte – scheduling



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	2	0.5	3	0.5	10	100
RB	t3	1	0.25	1.5	1	10	100
	t2	1	0.25	1.5	1	10	101

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

Events of task t1: WAIT at 1,5; WAKEUP after 1,0;

Simulare l'evoluzione del sistema per **3 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	wait	T1	TRUE		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	2	6	2	T3	100.75		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T3	1	0.5	3	1	10	100
RB	T2	1	0.5	3	1	10	101
WAITING	T1	2				11.5	100.75

$$T1 \rightarrow VRT = 100 + 1.5 * 0.5 = 100.75$$

RESCHED:  $100.75 + 1 * 0.5 = 101.25 < 101 \Rightarrow \text{FALSE} \Rightarrow$  ANCHE SE IL VRT DI T1 è MINORE, CONTINUA L'ESECUZIONE T3

EVENTO		TIME	TYPE	CONTEXT	RESCHED	T3 -> VRT = $100 + 1 * 1 = 101$ T1 -> VRT = 100.75	
		2.5	wake up	T3	FALSE		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	3	6	4	T3	101		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T3	1	0.25	1.5	1	11	101
RB	T1	2	0.5	3	0.5	11.5	100.75
	T2	1	0.25	1.5	1	10	101
WAITING							

EVENTO		TIME	TYPE	CONTEXT	RESCHED	T1 -> VRT = $100.75 + 3 * 0.5 = 102.25$	
		5.5	s QdT	T1	TRUE		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	3	6	4	T3	101		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T3	1	0.25	1.5	1	11	101
RB	T2	1	0.25	1.5	1	10	101
	T1	2	0.5	3	0.5	14.5	102.25
WAITING							

Valutazione della necessità di rescheduling per l'evento di WAKEUP:

Tempo dell'evento considerato: \_\_\_\_\_

Calcolo: \_\_\_\_\_

## esercizio n. 3 – memoria e file system

### prima parte – memoria

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

**MAXFREE = 3                      MINFREE = 2**

Si consideri la seguente **situazione iniziale**:

PROCESSO: P

\*\*\*\*\*

VMA: ...

PT: <c0 :1 R> <c1 :- -> <s0 :5 R> <s1 :- -> <d0 :7 R> <d1 :- ->  
<d2 :3 W> <d3 :- -> <p0 :6 W> <p1 :s2 R> <p2 :- ->

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

PROCESSO: Q ...

MEMORIA FISICA (pagine libere: 3)			
00 : <ZP>	01 : Pc0/Qc0/<X,0>		
02 : Qp0 D	03 : Pd2		
04 : ----	05 : Ps0/Qs0/<X,2>		
06 : Pp0	07 : Pd0		
08 : ----	09 : ----		

STATO del TLB			
Pc0 : 01 - 0: 1:	Pp0 : 06 - 1: 0:		
Pd2 : 03 - 1: 0:	-----		
Ps0 : 05 - 0: 1:	Pd0 : 07 - 0: 1:		
-----	-----		

SWAP FILE: Pd0 , Qd0 , Pp1/Qp1 , ----, ----, ----,

LRU ACTIVE: PD0, PS0, PC0,

LRU INACTIVE: pd2, pp0, qs0, qp0, qc0,

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 – read (Ps1), write(Pd0)**

PT del processo: P				
c0: <:1 R>	s0: <:5 R>	s1: <:- ->	p0: <:6 W>	p2: <:- ->

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: Qp0 (D)	03: Pd2
04: Ps1 / <X, 3>	05: Ps0 / Qs0 / <X, 2>
06: Pp0	07: Pd0
08: ----	09: ----

SWAP FILE	
s0: ----	s1: Qd0
s2: Pp1 / Qp1	s3: ----
s4: ----	s5: ----

Active: PS1, PD0, PS0, PC0Inactive: pd2, pp0, qs0, qp0, qc0**evento 2 – write (Ps0)**

PT del processo: P				
c0: <:1 R>	s0: <:2 W>	s1: <:4 R>	p0: <:s3 W>	p2: <:- ->

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: Ps0	03: Pd2
04: Ps1 / <X, 3>	05: Qs0 / <X, 2> (D)
06: ----	07: Pd0
08:	09:

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

Inactive: pd2, qs0, qc0

## seconda parte – memoria e file system

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

**MAXFREE = 3      MINFREE = 2**

Si consideri la seguente **situazione iniziale**:

MEMORIA FISICA (pagine libere: 5)			
00 : <ZP>		01 : Pc0 / <X, 0>	
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**. La pagina in cima alla pila è **Pp0**.

**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), read(fd, 8000)

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

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

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

eventi 3 e 4 – `fork(R)`, `lseek (fd, -5000)`, `write (fd, 10)`

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Rc0 / <X, 0>
02: Rp0 (D)	03: <F, 0> (D)
04: <F, 1>	05: Pp0
06:	07:

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F	3010	2	2	0

eventi 5-8 – `fd1 = open (G)`, `write(fd1, 4000)`, `close(fd)`, `close(fd1)`

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Rc0 / <X, 0>
02: Rp0 (D)	03: <G, 0>
04: ----	05: Pp0
06:	07:

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F	3010	1	2	1
file G	----	0	1	1

eventi 9 e 10 - `context switch(R)`, `write(fd, 100)`

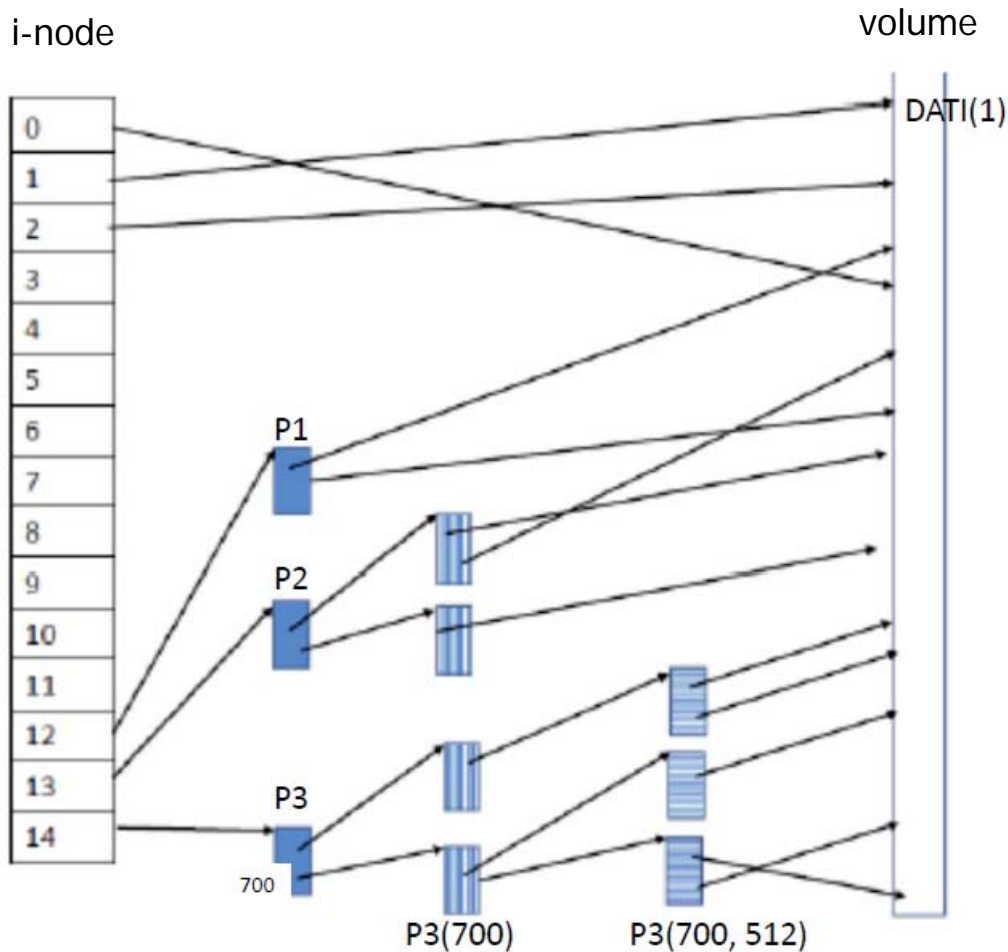
	f_pos	f_count	numero pagine lette	numero pagine scritte
file F	3110	1	3	1
file G	----	0	1	1

## esercizio n. 4 – Domanda

Si consideri il FS ext2 con dimensione di blocco = dim. pagina = 4Kbyte.  
Ogni puntatore occupa 4 byte.

Con riferimento alla figura seguente si consideri la seguente notazione:

- DATI(N) indica la pagina dati in posizione N; DATI(0) è la prima pagina dati del file
- P1, P2 e P3 sono i 3 blocchi contenenti puntatori di indirezione semplice
- $P_i(j)$  indica un blocco di puntatori al secondo livello di indirezione raggiunto dal puntatore numero j del blocco i di indirezione semplice (esempio in figura: P3(700) è il blocco puntato dal puntatore in posizione 700 (il 701-esimo, perché i puntatori sono numerati da 0) del blocco P3)
- $P_i(j,k)$  estende la stessa notazione ai blocchi di terzo livello – tripla indirezione – come P3(700,512) in figura, puntato dal puntatore in posizione 512 del blocco P3(700).



Indirizzamento di P1  $\Rightarrow 12 + 1024 - 1 = 1035$

Indirizzamento di P2(0)  $\Rightarrow 1036 - \dots - 1036 + 1024 - 1 = 2059$

Si supponga che un programma esegua in sequenza le seguenti operazioni su un file F:

1. open (la open non legge il file, ma solo l'i-node),
2. lseek(FP) – si posiziona all'inizio della **pagina** di numero FP, cioè DATI(FP)
3. read(NUM) – NUM è il numero di **pagine** lette

Si considerino i quattro casi riportati nella tabella sottostante (ciascun caso è indipendente dagli altri). Ripor-  
tare, per ogni caso richiesto, i blocchi dati e i blocchi puntatore a cui si deve accedere, e indicare il numero  
totale di blocchi dati e di blocchi puntatore trasferiti dal disco in memoria. Il primo caso è già compilato co-  
me esempio.

	1035 - 1036 -1037	2059 - 2060	2058 - 2059 -2060
FP=11, NUM=2	FP=1035, NUM=3	FP=2059, NUM=2	FP=2058, NUM=3
DATI(11)	P1	P2	P2
P1	DATI(1035)	P2(0)	P2(0)
DATI(12)	P2	DATI(2059)	DATI(2058)
	P2(0)	P2(1)	DATI(2059)
	DATI(1036)	DATI(2060)	P2(1)
	DATI(1037)		DATI(2060)
Numero Totale di trasferimenti da disco nei diversi casi			
3	6	5	6