Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria

prof.ssa Anna Antola **prof.ssa Donatella Sciuto**
prof. Luca Breveglieri **prof.ssa Cristina Silvano**

AXO – Architettura dei Calcolatori e Sistemi Operativi

SECONDA PARTE – giovedì 20 febbraio 2020

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 (6 punti) _____

esercizio 4 (1 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 solid, liquid
sem_t gas
int global = 0
```

```
void * soft (void * arg) {
   mutex_lock (&solid)
      sem_wait (&gas)
   mutex_unlock (&solid)                          /* statement A */
   global = 1
   mutex_lock (&liquid)
      sem_post (&gas)                             /* statement B */
   mutex_unlock (&liquid)
   return NULL
} /* end soft */
```

```
void * hard (void * arg) {
   mutex_lock (&liquid)
      sem_post (&gas)
      global = 2                                  /* statement C */
      sem_wait (&gas)
   mutex_unlock (&liquid)
   mutex_lock (&solid)
      sem_post (&gas)
   mutex_unlock (&solid)                          /* statement D */
   return NULL
} /* end hard */
```

```
void main ( ) {
   pthread_t th_1, th_2
   sem_init (&gas, 0, 0)
   create (&th_1, NULL, soft, NULL)
   create (&th_2, NULL, hard, 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	***thread***	
	th_1 – soft	**th_2** – hard
subito dopo stat. **A**	Esiste	Può esistere
subito dopo stat. **C**	Esiste	Esiste
subito dopo stat. **D**	Può esistere	Esiste
subito dopo stat. **E**	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**		
	solid	*liquid*	*gas*
subito dopo stat. **A**	0	0 - 1	0
subito dopo stat. **B**	0	1	1
subito dopo stat. **C**	1 - 0	1	1 - 0
subito dopo stat. **E**	0	0	1

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** – soft	**th_2** – hard	*global*
1	sem_wait(&gas)	mutex_lock(&solid)	2
2	mutex_lock(&liquid)	sem_wait(&gas)	1 - 2
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
      execl ("/acso/prog_x", "prog_x", NULL)
      exit (-1)
   } else {                   // codice eseguito da P
      read (stdin, msg, 5)
      pid = wait (&status)
   } // end_if pid1
   exit (0)
} // prova
```

```
// programma prog_x.c
// dichiarazione e inizializzazione dei mutex presenti nel codice
// dichiarazione dei semafori presenti nel codice
void * me (void * arg) {
   sem_post (&busy)
   mutex_lock (&lonely)
      sem_wait (&busy)
   mutex_unlock (&lonely)
   return NULL
} // me

void * you (void * arg) {
   nanosleep (2)
   mutex_lock (&lonely)
      sem_wait (&busy)
   mutex_unlock (&lonely)
   sem_post (&busy)
   return NULL
} // you

main ( ) {  // codice eseguito da Q
   pthread_t th_1, th_2
   sem_init (&busy, 0, 0)
   create (&th_1, NULL, me, NULL)
   create (&th_2, NULL, you, NULL)
   pid = fork ( )             // Q crea R
   if (pid == 0) {            // codice eseguito da R
      read (stdin, msg, 24)
      exit (-1)
   } else {                   // codice eseguito da Q
      join (th_2, NULL)
      join (th_1, NULL)
   } // if pid
   exit (1)
} // main
```

Un processo ***P*** esegue il programma ***prova*** e crea un processo figlio ***Q*** che esegue una mutazione di codice (programma ***prog_x***). La mutazione di codice va a buon fine e ***Q*** crea i thread ***th_1*** e ***th_2***, e 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. Si completi la tabella riportando quanto segue:

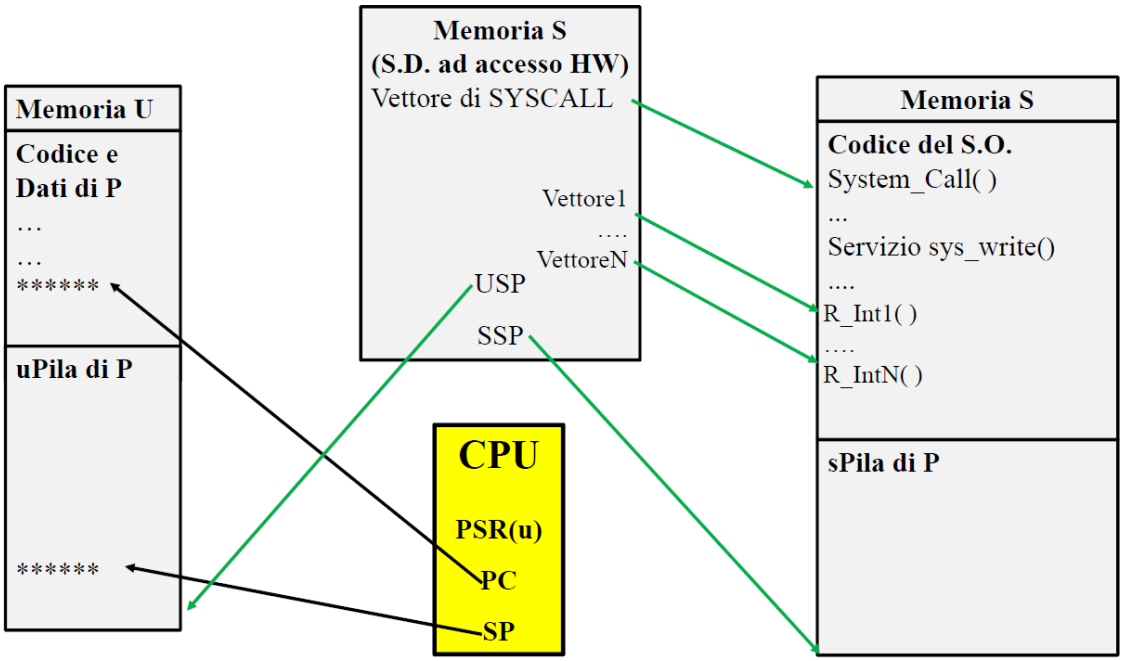
- ⟨ *PID*, *TGID* ⟩ di ciascun processo che viene creato
- ⟨ *identificativo del processo-chiamata di sistema / libreria* ⟩ nella prima colonna, dove necessario e in funzione del codice proposto
- in ciascuna riga lo stato dei processi **al termine dell'evento o della chiamata associata alla riga stessa**; si noti che la prima riga della tabella **potrebbe essere solo parzialmente completata**

TABELLA DA COMPILARE (numero di colonne non significativo)

identificativo simbolico del processo		**IDLE**	**P**	**Q**	**th_1**	TH2	R	
evento oppure *processo-chiamata*	*PID*	**1**	**2**	3	4	5	6	
	TGID	**1**	**2**	3	3	3	4	
Q –create th_2 (*) th_1 è pronto già da tempo	**0**	**pronto**	**A read da stdin**	**esec**	**pronto (*)**	pronto	NE	
interrupt da RT_clock e scadenza quanto di tempo	1	pronto	A read	pronto	ESEC	pronto	NE	
TH1 - sem_post(&busy)	2	pronto	A read	pronto	ESEC	pronto	NE	
5 interrupt da std_in, tutti i 5 caratteri richiesti trasferiti	3	pronto	ESEC	pronto	pronto	pronto	NE	
P - wait	4	pronto	A	pronto	pronto	esec	NE	
TH2 - nanosleep	5	pronto	A	esec	pronto	A	NE	
Q - fork	6	pronto	A wait	ESEC	pronto	A nano	pronto	
Q - join(TH2)	7	pronto	A wait	A join	ESEC	A nano	pronto	
interrupt da RT_clock e scadenza quanto di tempo	8	pronto	A wait	A join	pronto	A nano	ESEC	
R - read	9	pronto	A	A	esec	A	A	
TH1 - mutex_lock(&lonely)	10	pronto	A wait	A join	ESEC	A nano	A read	
Interrupt da RT_clock e scadenza del timeout	11	pronto	A	A	pronto	esec	A	
TH2 - mutex_lock(&lonely)	12	pronto	A wait	A join	ESEC	A lock	A read	

seconda parte – moduli, pila e strutture dati HW

Si consideri un processo **P** in esecuzione in modo U della funzione *main.* La figura sotto riportata e i valori nella tabella successiva descrivono compiutamente, ai fini dell'esercizio, il contesto di **P**.



Un processo **Q** è in attesa di un evento. I processi **P** e **Q** sono gli unici di interesse nel sistema

Si assuma che i valori della situazione iniziale di interesse siano i seguenti:

processo P	
PC	X
SP	Y
SSP	Z
USP	W
descrittore di P.stato	PRONTO

// è all'interno di *main*

RUNQUEUE	
CURR	P
RB.LFT	NULL

Si consideri la seguente **serie di eventi**.

evento A&B

A. Durante l'esecuzione del codice utente si verifica **Interrupt_1** che manda in esecuzione la routine di risposta all'interrupt **R_int_1**.

B. Durante l'esecuzione della routine di risposta R_int_1 si verifica **Interrupt_2** che viene accettato e manda in esecuzione la routine di risposta all'interrupt **R_int_2**.

Completare le tabelle seguenti con i valori assunti dagli elementi subito dopo il verificarsi di evento **A&B**.

processo P	
PC	// non di interesse
SP	K
SSP	Z
USP	W
descrittore di P.stato	ESEC (S)

Perchè "PRONTO"?

(oppure Z - 4)

(K)

(Z)

sPila di P
PSR (S)
Rientro a R_int_1 da R_int_2
PSR (U)
Rientro al codice utente da R_int_1

RUNQUEUE	
CURR	P
RB.LFT	NULL

evento C

La routine di risposta all'interrupt **R_int_2** risveglia il processo **Q** che viene portato in stato di pronto. Il processo **Q** ha maggiori diritti di esecuzione di **P**.

Completare le tabelle seguenti con i valori assunti dagli elementi **subito dopo l'esecuzione dell'istruzione IRET che termina** la routine di risposta all'interrupt **R_int_2**.

processo P	
PC	// non di interesse
SP	K
SSP	Z
USP	W
descrittore di P.stato	ESEC (S)

Perchè "PRONTO"?

(K)

(Z)

sPila di P
PSR (U)
Rientro al codice utente d R_int_1

RUNQUEUE	
CURR	P
RB.LFT	Q

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 = 1**

situazione iniziale (esistono un processo P e un processo 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>
          P  7FFFFFFC,  3 , W  ,  P  ,  A  , <-1, 0>

    PT : <c0 :1 R>  <s0 :4 R>  <s1 :- ->  <d0 :5 R>  <d1 :- ->
         <p0 :2 R>  <p1 :7 W>  <p2 :- ->

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

PROCESSO: Q  ***** non di interesse per l'esercizio ******************

____MEMORIA FISICA____(pagine libere: 3)_____________________________
      00 : <ZP>                 ||  01 : Pc0 / Qc0 / <X, 0>  ||
      02 : Pp0 / Qp0            ||  03 : ----                ||
      04 : Ps0 / Qs0            ||  05 : Pd0 / Qd0           ||
      06 : Qp1  D               ||  07 : Pp1                 ||
      08 : ----                 ||  09 : ----                ||

____STATO del TLB_____________________________________________________
      Pc0 : 01 -  0: 1:         ||   Pp0 : 02 -  1: 0:       ||
      Ps0 : 04 -  1: 0:         ||   Pd0 : 05 -  1: 0:       ||
      Pp1 : 07 -  1: 1:         ||         -----             ||
            -----               ||         -----             ||

SWAP FILE:    ----, ----, ----, ----, ----, ----,
LRU ACTIVE:   PC0,  PP1,
LRU INACTIVE: pp0,  pd0,  ps0,  qp1,  qd0,  qs0,  qp0,  qc0,
```

evento 1: *write* (Pp2, Pp3)

PT del processo: **P**				
p0: :2 R	p1: :7 W	p2: :3 W	p3: :8 W	p4: :- -

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: Pp0 / Qp0	03: Pp2
04: Ps0 / Qs0	05: Pd0 / Qd0
06: Qp1 (D)	07: Pp1
08: Pp3	09: - - - -

LRU ACTIVE: PP3, PP2, PC0, PP1

LRU INACTIVE: pp0, pd0, ps0, qp1, qd0, qs0, qp0, qc0

evento 2: *mmap* (0x000050000000, 3, W, S, M, "F", 2)

evento 3: *write* (Pp4)

VMA del processo **P**							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
M	0x 0000 5000 0	3	W	S	M	F	2
P	0x 7FFF FFFF 9	6	W	P	A	-1	0

PT del processo: **P**				
s0: :s1 R	s1: :- -	d0: :s2 R	d1: :- -	p0: :2 R
p1: :7 W	p2: :3 W	p3: :8 W	p4: :4 W	p5: :- -
m00: :- -	m01: :- -	m02: :- -		

process P – NPV of PC and SP: c0 p4

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: Pp0 / Qp0	03: Pp2
04: Pp4	05: - - - -
06: - - - -	07: Pp1
08: Pp3	09: - - - -

SWAP FILE	
s0: Qp1	s1: Ps0 / Qs0
s2: Pd0 / Qd0	s3:
s4:	s5:

LRU ACTIVE: PP4, PP3, PP2, PC0, PP1

LRU INACTIVE: pp0, qp0, qc0

evento 4: *write* (Pm01)

PT del processo: **P**				
m00: :- -	m01: :5 W	m02: :- -		

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: Pp0 / Qp0	03: Pp2
04: Pp4	05: Pm01 / <F, 3>
06: - - - -	07: Pp1
08: Pp3	09: - - - -

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: c0, p0

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

```
__MEMORIA FISICA____(pagine libere: 1)_____________________________
     00 : <ZP>                 ||  01 : Pc0 / Qc0 / <X, 0>  ||
     02 : Pp0 / Qp0            ||  03 : Qp1  D              ||
     04 : Pp1                  ||  05 : <F, 0>              ||
     06 : <F, 1>  D            ||  07 : ----                ||
```

processo	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P	**F**	**6500**	**1**	**2**	**0**

ATTENZIONE: **è presente la colonna "processo" in cui va specificato il nome del processo a cui si riferiscono le informazioni "f_pos" e "f_count" (campi di struct file) relative al file indicato**

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 *f_count* diventa = 0.

È in esecuzione il processo **P**.

Per ciascuno degli eventi seguenti, 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.

eventi 1 e 2: *write* (fd1, 5500), *read* (fd1, 4000)

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: Pp0 / Qp0	03: Qp1 (D)
04: Pp1	05: <F, 2> (D)
06: <F, 3>	07:

processo	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P	**F**	16000	1	4	1

6500 => 6500 + 5500 = 12000

12000 => 12000 + 4000 => 16000

0 ---- 4096 ---- 8192 ---- 12288 ---- 16384 ---- 20480 ---- 24576
0 1 2 3 4 5

evento 3: *context_switch* (“Q”)

NOTA BENE: la pagina p0 di P (condivisa con Q) e la pagina p1 di P risultano marcate dirty nel TLB, che non è mostrato

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: Pp0 / Qp0 (D)	03: Qp1 (D)
04: Pp1 (D)	05: <F, 2> (D)
06: <F, 3>	07:

eventi 4 e 5: fd2 = *open* (“F”), *lseek* (fd2, 13000)

processo	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P	**F**	16000	1	4	1
Q	**F**	13000	1		

evento 6: *write* (fd2, 100)

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: Pp0 / Qp0 (D)	03: Qp1 (D)
04: Pp1 (D)	05: <F, 2> (D)
06: <F, 3> (D)	07:

processo	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P	**F**	16000	1	4	1
Q	**F**	13100			

evento 7: *close* (fd2)

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <X, 0>
02: Pp0 / Qp0 (D)	03: Qp1 (D)
04: Pp1 (D)	05: <F, 2>
06: <F, 3>	07:

processo	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P	**F**	16000	1	4	3
Q	**F**	- - - -	0		

esercizio n. 4 – 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	1	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	256	W	P	A	-1	0
M1	0000 3000 0	3	W	P	M	F	2
P	7FFF FFFF A	5	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
M1	0000 3000 0	0	0	384	0
P	7FFF FFFF A	255	511	511	506

2. Numero di pagine necessarie

pag PGD: 1

pag PUD: 2

pag PMD: 2

pag PT: 4

pag totali: 9

3. Numero di pagine virtuali occupate dal processo: 270

4. Rapporto di occupazione: 9 / 270 = 3.33%

5. Dimensione massima del processo in pagine virtuali: 512 * 4 = 2048

6. Rapporto di occupazione con dimensione massima: 9 / 2048 = 0.43294%

spazio libero per brutta copia o continuazione

spazio libero per brutta copia o continuazione