Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria

prof. Luca Breveglieri **prof.ssa Donatella Sciuto**
prof. Gerardo Pelosi **prof.ssa Cristina Silvano**

AXO – Architettura dei Calcolatori e Sistemi Operativi

SECONDA PARTE – giovedì 23 giugno 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 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 (4 punti) _____

esercizio 4 (3 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 middle
sem_t front, back
int global = 0
```

```
void * wake (void * arg) {
   mutex_lock (&middle)
   sem_post (&front)
   global = 1                                          /* statement A */
   mutex_unlock (&middle)
   global = 2
   sem_wait (&back)
   mutex_lock (&middle)
   sem_wait (&back)
   mutex_unlock (&middle)
   return (void *) 3
} /* end wake */
```

```
void * sleep (void * arg) {
   mutex_lock (&middle)
   global = 4                                          /* statement B */
   sem_wait (&front)
   mutex_unlock (&middle)
   mutex_lock (&middle)
   sem_post (&back)
   global = 5                                          /* statement C */
   mutex_unlock (&middle)
   global = 6
   return NULL
} /* end sleep */
```

```
void main ( ) {
   pthread_t th_1, th_2
   sem_init (&front, 0, 0)
   sem_init (&back, 0, 1)
   create (&th_2, NULL, sleep, NULL)
   create (&th_1, NULL, wake, 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	***thread***	
	th_1 – *wake*	**th_2 – *sleep***
subito dopo stat. **A**	Esiste	Esiste
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**			
	middle	*front*	*back*	*global*
subito dopo stat. **A**	1	1	1	1
subito dopo stat. **B**	1	1 / 0	1 / 0	4 / 2
subito dopo stat. **C**	1	0	2 / 1	5 / 2
subito dopo stat. **D**	0	0	0	3 / 6

Il sistema può andare in stallo (*deadlock*), con uno o più *thread* che si bloccano, in (almeno) **due 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 – *wake***	**th_2 – *sleep***	*global*
1	mutex_lock(&middle)	sem_wait(&front)	4
2	sem_wait(&back)	mutex_lock(&middle)	2 / 4
3			

esercizio n. 2 – processi e nucleo

prima parte – gestione dei processi

```
// programma fb1.c
int main ( ) {
      pid1 = fork ( )                          // creazione del processo Q
      if (pid1 == 0) {                         // codice eseguito da Q
         execl ("/acso/fb2", "fb2", NULL)
         exit (-2)
      } /* if */
      pid2 = fork ( )                          // creazione del processo R
      if (pid2 == 0) {                         // codice eseguito da R
         execl ("/acso/fb2", "fb2", NULL)
      } /* if */
      pid = wait (NULL)
} /* main */
// programma fb2.c
int main ( ) {
      pid1 = fork ( )                          // creazione del processo S
      if (pid1 == 0) {                         // codice eseguito da S
         write (stdout, "proc S", 6)
         execl ("/acso/fb3", "fb3", NULL)
         exit (-2)
      } /* if */
      pid2 = waitpid (pid1, NULL, 0)           // codice eseguito da R
} /* main */
// programma fb3.c
int main ( ) {
      pid1 = fork ( )                          // creazione del processo T
      write (stdout, "both proc", 9)
      if (pid1 == 0) {
         write (stdout, "proc T", 6)
         exit (-2)
      } /* if */
      pid2 = waitpid (pid1, NULL, 0)           // codice eseguito da S
} /* main */
```

Un processo **P** esegue il programma **fb1.c** e crea i processi figli **Q** e **R.** Il processo **Q** esegue una mutazione di codice che **non** va a buon fine. Il processo **R** effettua con successo una mutazione di codice, esegue il programma **fb2.c** e crea il processo **S**. Il processo **S** effettua con successo una mutazione di codice, esegue il programma **fb3.c** e crea il processo **T**.

Si simuli l'esecuzione dei processi così come risulta dal codice dato, dagli eventi indicati.

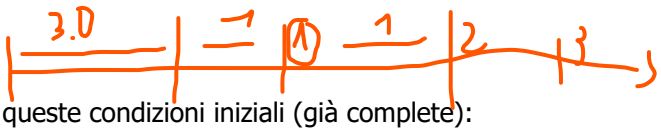
Si completi la tabella riportando quanto segue:

- PID e TGID di ogni processo che viene creato
- identificativo del processo-chiamata di sistema / libreria nella prima colonna, dove necessario
- in funzione del codice proposto in ciascuna riga, lo stato dei processi **al termine del tempo indicato**

TABELLA DA COMPILARE (numero di colonne non significativo)

identificativo simbolico del processo		**IDLE**	**P**	**Q**	**R**	**S**	**T**
evento oppure processo-chiamata	**PID**	**1**	2	3	4	5	6
	TGID	**1**	2	3	4	5	6
P – fork	**1**	**pronto**	**esec**	**pronto**	**NE**	**NE**	**NE**
P - fork	**2**	pronto	ESEC	pronto	pronto	NE	NE
P - wait(NULL)	**3**	pronto	A wait	ESEC	pronto	NE	NE
Q - execl	**4**	pronto	A wait	ESEC	pronto	NE	NE
Q - exit	**5**	pronto	ESEC	NE	pronto	NE	NE
P - exit	**6**	pronto	NE	NE	ESEC	NE	NE
R - execl	**7**	pronto	NE	NE	ESEC	NE	NE
interrupt da RT_clock (scadenza qdt)	**8**	pronto	NE	NE	ESEC	NE	NE
R - fork	**9**	pronto	NE	NE	ESEC	pronto	NE
R - wait	**10**	pronto	NE	NE	A wait	ESEC	NE
S - write	**11**	ESEC	NE	NE	A wait	A write	NE
Interrupt da STD_OUT, tutti i blocchi scritti	**12**	**pronto**	**NE**	**NE**	**attesa (wait S)**	**esec**	**NE**
S - execl	**13**	pronto	NE	NE	A wait	ESEC	NE
S - fork	**14**	pronto	NE	NE	A wait	ESEC	pronto
S - write	**15**	pronto	NE	NE	A wait	A write	ESEC
T - write	**16**	ESEC	NE	NE	A wait	A write	A write

seconda parte – scheduling



Si consideri uno scheduler CFS con **due task** caratterizzato da queste condizioni iniziali (già complete):

CONDIZIONI INIZIALI (già complete)							
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
	2	6	2	T1	100		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	T1	1	0,5	3	1	10	100
RB	T2	1	0,5	3	1	20	103

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

```
Events of task T1:   EXIT at 5.0
Events of task T2:   WAIT at 1.0
                     WAKEUP after 1.0
```

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**	
	3.0	S.Q.d.T	T1	TRUE	
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**
	2	6	2	T2	103

T1 -> VRT = 100 + 3 * 1 = 103

TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	T2	1	0.5	3	1	20	103
RB	T1	1	0.5	3	1	13	103
WAITING							

EVENTO 2	**TIME**	**TYPE**	**CONTEXT**	**RESCHED**	
	4.0	wait	T2	TRUE	
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**
	1	6	1	T1	103

T2 -> VRT = 103 + 1 * 1= 104

TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	T1	1	1	6	1	13	103
RB							
WAITING	T2	1				21	104

T1 -> VRT = 103 + 1 * 1 = 104

T2 -> VRT = 104

EVENTO 3		TIME	TYPE	CONTEXT	RESCHED		
		5.0	WAKE	T1	FALSE		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	2	6	2	T1	104		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T1	1	0.5	1	1	14	104
RB	T2	1	0.5	1	1	21	104
WAITING							

T1 -> VRT = 104 + 1 * 1 = 105

EVENTO 4		TIME	TYPE	CONTEXT	RESCHED		
		6.0	EXIT	T1	TRUE		
RUNQUEUE	NRT	PER	RQL	CURR	VMIN		
	1	6	1	T2	104		
TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT	T2	1	1	6	1	21	104
RB							
WAITING							

Valutazione della condizione di rescheduling alla **WAKEUP**

104 + 1 * 0.5 = 104 > 104 => FALSE

esercizio n. 3 – memoria virtuale

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 - NPV of PC and SP: c1, p0

```
PROCESSO: P  ************************************************************
    VMA : C  000000400,  2 , R  ,  P  ,  M  , <YY, 0>
          K  000000600,  1 , R  ,  P  ,  M  , <YY, 2>
          S  000000601,  1 , W  ,  P  ,  M  , <YY, 3>
          M0 000010000,  4 , W  ,  S  ,  M  , <F, 0>
          P  7FFFFFFFC,  3 , W  ,  P  ,  A  , <-1, 0>

    PT: <c0 :- ->  <c1 :1  R>  <k0 :- ->   <s0 :- ->  <p0 :2  W>  <p1 :- ->
        <p2 :- ->  <m00:3  W>  <m01:4  W>  <m02:5  W> <m03:- ->

    ___MEMORIA FISICA____(pagine libere: 4)_____________________
      00 : <ZP>                  ||  01 : Pc1 / <YY, 1>       ||
      02 : Pp0                   ||  03 : Pm00 / <F, 0>       ||
      04 : Pm01 / <F,1>          ||  05 : Pm02 / <F, 2>       ||
      06 : ----                  ||  07 : ----                ||
      08 : ----                  ||  09 : ----                ||

    ____STATO del TLB_____________________________________________
      Pc1  : 01 -  0: 1:         ||   Pp0  : 02 -  1: 1:      ||
      Pm00 : 03 -  1: 1:         ||   Pm01 : 04 -  1: 1:      ||
      Pm02 : 05 -  1: 1:         ||         -----             ||
            -----                ||         -----             ||
            -----                ||         -----             ||
            -----                ||         -----             ||

SWAP FILE:     ----, ----, ----, ----, ----, ----,
LRU ACTIVE:    PM02, PM01, PM00,  PP0,  PC1,
LRU INACTIVE:
```

eventi 1: *read* (Pc1) – *write* (Pp1) – 4 *kswapd*

MEMORIA FISICA	
00: <ZP>	01: Pc1 / <YY, 1>
02: Pp0	03: Pm00 / <F, 0>
04: Pm01 / <F, 1>	05: Pm02 / <F, 2>
06: Pp1	07:
08:	09:

LRU ACTIVE: PP1, PC1

LRU INACTIVE: pm02, pm01, pm00, pp0

eventi 2: *read* (Pc1) – *write* (Pp2, Pp3) – 4 *kswapd*

PT del processo: **P**				
p0: :s0 W	p1: :6 W	p2: :7 W	p3: :2 W	p4: - -

MEMORIA FISICA	
00: <ZP>	01: Pc1 / <YY, 1>
02: Pp3	03: - - - -
04: Pm01 / <F, 1>	05: Pm02 / <F, 2>
06: Pp1	07: Pp2
08:	09:

SWAP FILE	
s0: Pp0	s1:
s2:	s3:
s4:	s5:

LRU ACTIVE: PP3, PP2, PC1

LRU INACTIVE: pp1, pm02, pm01

eventi 3: *read* (Pc1) – *write* (Pm01) – 4 *kswapd*

LRU ACTIVE: PM01, PC1

LRU INACTIVE: pp3, pp2, pp1, pm02

eventi 4: *fork* (Q) – *context switch* (Q)

processo Q	NPV of **PC**: c1	NPV of **SP**: p3

PT del processo: **Q**				
p0: :s0 W	p1: :6 R	p2: :7 R	p3: :2 W	p4: - -

MEMORIA FISICA	
00: <ZP>	01: Pc1 / Qc1 / <YY, 1>
02: Qp3 (D)	03: Pp3 (D)
04: Pm01 / Qm01 / <F, 1>	05: Pm02 / Qm02 / <F, 2>
06: Pp1 / Qp1 (D)	07: Pp2 / Qp2 (D)
08:	09:

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

esercizio n. 4 – file system

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.**

```
processo P - NPV of PC and SP:  c1, p0

PROCESSO: P  ************************************************
    VMA : C  000000400,  2 , R  ,  P  ,  M  , <XX, 0>
          S  000000600,  2 , W  ,  P  ,  M  , <XX, 2>
          D  000000602,  2 , W  ,  P  ,  A  , <-1, 0>
          P  7FFFFFFFC,  3 , W  ,  P  ,  A  , <-1, 0>

    PT: <c0 :3  R>  <c1 :4  R>  <s0 :1  R>  <s1 :- ->  <d0 :- ->
        <d1 :- ->   <p0 :2  W>  <p1 :- ->   <p2 :- ->

____MEMORIA FISICA____(pagine libere: 3)_____________________________
      00 : <ZP>                  ||  01 : Ps0 / <XX, 2>          ||
      02 : Pp0 D                 ||  03 : Pc0 / <XX, 0>          ||
      04 : Pc1 / <XX, 1>         ||  05 : ----                   ||
      06 : ----                  ||  07 : ----                   ||

____STATO del TLB_____________________________________________________
      Ps0 : 01 -  0: 0:          ||  Pp0 : 02 -  1: 1:           ||
      Pc0 : 03 -  0: 0:          ||  Pc1 : 04 -  0: 1:           ||
            -----                ||        -----                 ||

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

LRU ACTIVE:   PC1, PP0,

LRU INACTIVE: pc0, ps0,
```

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

evento 1: *read* (fd1, 9000)

MEMORIA FISICA	
00: <ZP>	01: Ps0 / <XX, 2>
02: Pp0 (D)	03: Pc0 / <XX, 0>
04: Pc1 / <XX, 1>	05: <F, 2>
06: - - - -	07:

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P	F	9000	1	3	0

evento 2: *clone* (Q, c0)

VMA del processo **P/Q** (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

MEMORIA FISICA	
00: <ZP>	01: PQs0 / <XX, 2>
02: PQp0 (D)	03: PQc0 / <XX, 0>
04: PQc1 / <XX, 1>	05: <F, 2>
06: PQt00 (D)	07:

LRU ACTIVE: PQT00, PQC1, PQP0

LRU INACTIVE: pqc0, pqs0

evento 3: *context switch* (Q)

processo Q	NPV of **PC**: c1	NPV of **SP**: p0

TLB							
NPV	NPF	D	A	NPV	NPF	D	A
PQp0	2	1	1	PQs0	1	0	0

eventi 4: fd2 = *open* (G) – *write* (fd2, 4000)

MEMORIA FISICA	
00: <ZP>	01: <G, 0> (D)
02: PQp0 (D)	03: PQs0 / <XX, 2>
04: PQc1 / <XX, 1>	05:
06: PQt00	07:

SWAP FILE	
s0: PQs0	s1:

LRU ACTIVE: PQT00, PQC1, PQP0

LRU INACTIVE: pqc0

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P	F	9000	1	3	0
Q	G	4000	1	1	0

eventi 5: *context switch* (P) – *write* (fd1, 1000)

MEMORIA FISICA	
00: <ZP>	01: <G, 0> (D)
02: PQp0 (D)	03: PQs0 / <XX, 2>
04: PQc1 / <XX, 1>	05: <F, 2> (D)
06: PQt00	07:

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P	F	10000	1	4	0
Q	G	4000	1	1	0

eventi 6: *write* (fd1, 4000) – *close* (fd1) --- NB: *close* scrive su file le pagine dirty

MEMORIA FISICA	
00: <ZP>	01: <F, 3>
02: PQp0 (D)	03: PQs0 / <XX, 2>
04: PQc1 / <XX, 1>	05: - - - -
06: PQt00	07:

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P	F	- - - -	0	5	2
Q	G	4000	1	1	0

spazio libero per brutta copia o continuazione

spazio libero per brutta copia o continuazione