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 – mercoledì 1 luglio 2020

Cognome_____ **Nome** _____

Matricola_____ **Codice Persona**_____

Istruzioni – ESAME ONLINE

È vietato consultare libri, eserciziari e appunti, nonché cellulari e altri dispositivi mobili di calcolo o comunicazione. Chiunque non dovesse attenersi alla regola vedrà annullata la propria prova.

La prova va sempre consegnata completando la procedura prevista nel modulo (form) dell'esame con INVIO (SUBMIT) del testo risolto. Se lo studente intende RITIRARSI deve inviare messaggio di posta elettronica (email) al docente dopo avere completata la procedura.

Dallo **HONOR CODE**

In qualsiasi progetto o compito, gli studenti devono dichiarare onestamente il proprio contributo e devono indicare chiaramente le parti svolte da altri studenti o prese da fonti esterne.

Ogni studente garantisce che eseguirà di persona tutte le attività associate all'esame senza alcun aiuto di altri; la sostituzione di identità è un reato perseguibile per legge.

Durante un esame, gli studenti non possono accedere a fonti (libri, note, risorse online, ecc) diverse da quelle esplicitamente consentite.

Durante un esame, gli studenti non possono comunicare con nessun altro, né chiedere suggerimenti.

In caso di esame a distanza, gli studenti non cercano di violare le regole a causa del controllo limitato che il docente può esercitare.

L'accettazione dello Honor Code costituisce prerequisito per l'iscrizione agli esami.

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, come anche il prefisso pthread delle primitive di libreria NPTL):

```
pthread_mutex_t here
sem_t near, far
int global = 0
```

```
void * alpha (void * arg) {
   mutex_lock (&here)
      sem_wait (&far)
   mutex_unlock (&here)                          /* statement A */
   global = 1
   sem_wait (&near)
   mutex_lock (&here)
      sem_post (&near)                           /* statement B */
   mutex_unlock (&here)
   return NULL
} /* end alpha */
```

```
void * omega (void * arg) {
   mutex_lock (&here)
      sem_post (&far)
      global = 2                                 /* statement C */
      sem_wait (&near)
   mutex_unlock (&here)                          /* statement D */
   sem_wait (&near)
   return (void *) 3
} /* end omega */
```

```
void main ( ) {
   pthread_t th_1, th_2
   sem_init (&near, 0, 2)
   sem_init (&far,  0, 0)
   create (&th_1, NULL, alpha, NULL)
   create (&th_2, NULL, omega, NULL)
   join (th_2, &global)                          /* statement E */
   join (th_1, 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 – alpha	**th_2** – omega
subito dopo stat. **A**		
subito dopo stat. **C**		
subito dopo stat. **D**		
subito dopo stat. **E**		

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**		
	here	*near*	*far*
subito dopo stat. **A**			
subito dopo stat. **B**			
subito dopo stat. **C**			
subito dopo stat. **D**			

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** – alpha	**th_2** – omega	*global*
1			
2			
3			

esercizio n. 2 – processi e nucleo

prima parte – gestione dei processi

```
// programma prova.c
main ( ) {
   pid1 = fork ( )
   if (pid1 == 0) {                              // codice eseguito da Q
      execl ("/acso/prog_x", "prog_x", NULL)
      exit (-1)
   } /* if */
   pid2 = fork ( )
   fd = open ("/acso/esame", ORDWR)              // lettura di 2 blocchi
   if (pid2 == 0) {                              // codice eseguito da R
      write (fd, vett, 4096)                     // scrive 4 blocchi
      exit (-2)
   } else {
      pid1 = waitpid (pid1, &status, 0)
   } /* if */
   exit (0)
} /* prova */
```

```
// programma prog_x.c
// dichiarazione e inizializzazione dei mutex presenti nel codice
// dichiarazione dei semafori presenti nel codice
```

`void * soft(void * arg) {`		`void * hard (void * arg) {`
`sem_post (&gas)`		`mutex_lock (&liquid)`
`mutex_lock (&liquid)`		`sem_wait (&gas)`
`sem_wait (&gas)`		`mutex_unlock (&liquid)`
`mutex_unlock (&liquid)`		`sem_post (&gas)`
`return NULL`		`return NULL`
`} // soft`		`} // hard`

```
main ( ) {  // codice eseguito da Q
   pthread_t th_1, th_2
   sem_init (&gas, 0, 1)
     create (&th_1, NULL, soft, NULL)
     nanosleep (6)
     create (&th_2, NULL, hard, NULL)
     join (th_2, NULL)
     join (th_1, NULL)
     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***) e un figlio ***R***. La mutazione di codice va a buon fine e ***Q*** crea i thread ***th_1*** e ***th_2***.

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, e nell'ipotesi che ***th_1***, che esegue ***soft***, abbia già eseguito la ***mutex_lock*** ma non ancora la ***sem_wait***. 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**			
evento oppure *processo-chiamata*	*PID*	**1**	**2**					
	TGID	**1**	**2**					
P –pid2=fork ()	**0**	**pronto**	**esec**	**A nanosleep**	**pronto**			
	1							
	2							
2 interrupt da DMA_in, tutti i blocchi richiesti da open trasferiti	3							
	4	pronto	A	A	pronto	esec	NE	
	5							
interrupt da RT_clock e scadenza quanto di tempo	6							
	7	pronto	A	esec	pronto	A	NE	
	8							
	9	pronto	A	pronto	pronto	esec	pronto	
	10							
	11							
	12	pronto	A	pronto	NE	A	esec	

seconda parte – scheduler CFS

Si consideri uno Scheduler CFS con **2 task** caratterizzato da queste condizioni iniziali (**da completare**):

CONDIZIONI INIZIALI (da completare)							
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
	2	6		t1	100		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT	t1	1				10	101.0
RB	t2	1				30	100.5

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

```
Events of task t1:  CLONE at 0.5;  EXIT at 2.5;
Events of task t2:  WAIT at 0.5;   WAKEUP after 2.5;
```

Simulare l'evoluzione del sistema per **4 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**		
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT							
RB							
WAITING							

EVENTO		**TIME**	**TYPE**	**CONTEXT**	**RESCHED**		
RUNQUEUE	**NRT**	**PER**	**RQL**	**CURR**	**VMIN**		
TASK	**ID**	**LOAD**	**LC**	**Q**	**VRTC**	**SUM**	**VRT**
CURRENT							
RB							
WAITING							

EVENTO	TIME	TYPE	CONTEXT	RESCHED

RUNQUEUE	NRT	PER	RQL	CURR	VMIN

TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT							
RB							
WAITING							

EVENTO	TIME	TYPE	CONTEXT	RESCHED

RUNQUEUE	NRT	PER	RQL	CURR	VMIN

TASK	ID	LOAD	LC	Q	VRTC	SUM	VRT
CURRENT							
RB							
WAITING							

Condizioni di rescheduling a **clone** del **task t1**:

clone:

Condizioni di rescheduling a **wake_up** del **task t2**:

wake_up:

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 = 1 **MINFREE = 1**

Situazione iniziale (esiste un processo P):

```
PROCESSO: P
*********************************************************
    VMA :  C   000000400,  2,  R,  P,  M,  <XX, 0>
           M0  000010000,  1,  W,  S,  M,  <G,  2>
           P   7FFFFFFFC,  3,  W,  P,  A,  <-1, 0>

PT:  <c0:- ->  <c1:1 R>  <p0:2 W>  <p1:- ->  <p2:- ->  <m00:- ->

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:          ||
           -----                ||       -----                ||
           -----                ||       -----                ||
           -----                ||       -----                ||

LRU ACTIVE:   PP0, PC1,
LRU INACTIVE:
```

evento 1: ***mmap*** **(0x000030000000, 3, W, P, M, "F", 2),**
mmap **(0x000040000000, 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
M1							
M2							

evento 2: *read* (Pm20, Pm21, Pm11), *write* (Pm20, Pm00)

PT del processo: **P (completare con pagine di VMA)**				
c0: - -	c1: 1 R	p0: 2 W	p1: - -	p2: - -

MEMORIA FISICA	
00: <ZP>	01:
02:	03:
04:	05:
06:	07:

LRU ACTIVE: _____

LRU INACTIVE: _____

evento 3: *read* (Pc1, Pp0, Pm20), 4 *kswapd*

LRU ACTIVE: _____

LRU INACTIVE: _____

evento 4: *write* (Pm10)

MEMORIA FISICA	
00: <ZP>	01:
02:	03:
04:	05:
06:	07:

LRU ACTIVE: _____

LRU INACTIVE: _____

Indicare la decomposizione dell'indirizzo della prima pagina della VMA M0 nella TP:

PGD	PUD	PMD	PT

seconda parte – 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 : Pc2 / <X, 2>        ||
      02 : Pp0                 ||  03 : ----                ||
      04 : ----                ||  05 : ----                ||
      06 : ----                ||  07 : ----                ||

____STATO del TLB_____________________________________________________
     Pc2 : 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**.

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, 10000)

MEMORIA FISICA	
00: <ZP>	01:
02:	03:
04:	05:
06:	07:

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F				

eventi 3 e 4: *fork* (“Q”), *context switch* (“Q”)

MEMORIA FISICA	
00: <ZP>	01:
02:	03:
04:	05:
06:	07:

	f_pos	f_count	numero pagine lette	numero pagine scritte
file F				

evento 5: *write* (fd, 2000)

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

eventi 6 e 7: *close* (fd), *context switch* (“P”)

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

evento 8: *write* (fd, 16000)

MEMORIA FISICA	
00: <ZP>	01:
02:	03:
04:	05:
06:	07:

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

evento 9: *close* (fd)

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