Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria

prof.ssa Anna Antola — **prof. Giuseppe Pelagatti**
prof. Luca Breveglieri — **prof.ssa Donatella Sciuto**
prof. Roberto Negrini — **prof.ssa Cristina Silvano**

AXO – Architettura dei Calcolatori e Sistemi Operativi

Prova di giovedì 4 febbraio 2016

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 2 h : 15 m

Valore indicativo di domande ed esercizi, voti parziali e voto finale:

esercizio 1 (4 punti) _____

esercizio 2 (3 punti) _____

esercizio 3 (2 punti) _____

esercizio 4 (2 punti) _____

esercizio 5 (3 punti) _____

esercizio 6 (2 punti) _____

voto finale: (16 punti) _____

CON SOLUZIONI (in corsivo)

esercizio n. 1 – programmazione concorrente

Si consideri il programma C seguente (gli "#include" e le inizializzazioni dei mutex sono omessi):

```
pthread_mutex_t black, white
sem_t new, old
int global = 0
```

```
void * left (void * arg) {
   pthread_mutex_lock (&black)
   sem_post (&new)                                   /* statement A */
   global = 1
   pthread_mutex_unlock (&black)
   return NULL
} /* end left */
```

```
void * mid (void * arg) {
   pthread_mutex_lock (&black)
   sem_wait (&new)
   sem_wait (&old)
   pthread_mutex_lock (&white)                       /* statement B */
   pthread_mutex_unlock (&white)
   sem_post (&old)
   pthread_mutex_unlock (&black)
   return (int *) arg
} /* end mid */
```

```
void * right (void * arg) {
   pthread_mutex_lock (&white)                       /* statement C */
   sem_wait (&old)
   pthread_mutex_unlock (&white)
   global = 3
   return NULL
} /* end right */
```

```
void main ( ) {
   pthread_t th_1, th_2, th_3
   sem_init (&new, 0, 0)
   sem_init (&old, 0, 1)
   pthread_create (&th_2, NULL, mid, (void *) 2)
   pthread_create (&th_1, NULL, left, NULL)
   pthread_create (&th_3, NULL, right, NULL)
   pthread_join (th_1, NULL)
   pthread_join (th_2, &global)                      /* statement D */
   pthread_join (th_3, 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	**th_2**	**th_3**
subito dopo stat. **A**	*ESISTE*	*ESISTE*	*PUÒ ESISTERE*
subito dopo stat. **B**	*PUÒ ESISTERE*	*ESISTE*	*PUÒ ESISTERE*
subito dopo stat. **C**	*PUÒ ESISTERE*	*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)

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**		
	new	*old*	*global*
subito dopo stat. **A**	*1*	*0 / 1*	*0 / 3*
subito dopo stat. **B**	*0*	*0*	*1*
subito dopo stat. **D**	*0*	*0 / 1*	*2 / 3*

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, con il valore (o i valori) della variabile *global*:

caso	**th_1**	**th_2**	**th_3**	*global*
1	***lock black***	***wait new***		*0 / 3*
2		***lock white***	***wait old***	*1*
3		***wait old***		*1 / 3*

esercizio n. 2 – gestione dello stato 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 */
   fd = open ("/acso/esame",ORDWR)
   pid2 = fork ( )
   if (pid2 == 0) {   // codice eseguito da R
      write (fd, vett, 50)
      exit (-2)
   } else {
      pid1 = waitpid (pid1, &status, 0)
   } /* if */
   exit (0)
}  /* prova */
```

```
// programma prog_x.c
   pthread_mutex_t GATE = PTHREAD_MUTEX_INITIALIZER
   sem_t GO
```

```
void * FIRST (void * arg) {
   sem_wait (&GO)
   pthread_mutex_lock (&GATE)
   sem_post (&GO)
   pthread_mutex_unlock (&GATE)
   sem_wait (&GO)
   return NULL
}  /* FIRST */
```

```
void * LAST (void * arg) {
   pthread_mutex_lock (&GATE)
   sem_post (&GO)
   pthread_mutex_unlock (&GATE)
   sem_wait (&GO)
   sem_post (&GO)
   return NULL
}  /* LAST */
```

```
main ( ) {  // codice eseguito da S e Q
   pthread_t TH_1, TH_2
   sem_init (&GO, 0, 0)
   pthread_create (&TH_1, NULL, FIRST, NULL)
   pthread_create (&TH_2, NULL, LAST, NULL)
   pthread_join (TH_2, NULL)
   pthread_join (TH_1, NULL)
   exit (1)
}  /* main */
```

Un processo ***S*** esegue il programma ***prog_x*** e un processo ***P*** esegue il programma ***prova***. Il processo ***S*** crea i thread ***TH1*** e ***TH2***, mentre il processo ***P*** crea i processi ***Q*** e ***R***. Il processo ***Q*** esegue una mutazione di codice che non va a buon fine.

Si simuli l'esecuzione dei processi (fino a ***udt* = 200**) così come risulta dal codice dato, dagli eventi indicati e ipotizzando che il processo ***P*** non abbia ancora eseguito la prima *fork*. **Si completi** la tabella riportando quanto segue:

- ⟨ *PID*, *TGID* ⟩ di ogni 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 del tempo indicato**; si noti che la prima riga della tabella **potrebbe essere solo parzialmente completata**

NOTA BENE:

- si considerino **solo le funzioni marcate in grassetto** nel codice dato
- l'esecuzione della funzione *exec* **non sospende il processo**

TABELLA DA COMPILARE (numero di colonne non significativo)

identificativo simbolico del processo		***IDLE***	***S***	***P***	*TH1*	*Q*	*TH2*	*R*	
	PID	**1**	**2**	*3*	*4*	*5*	*6*	*7*	
evento processo-chiamata	*TGID*	**1**	**2**	*3*	*2*	*5*	*2*	*7*	
S – *sem_init*	**0**	**pronto**	**esec**	**pronto**	*NE*	*NE*	*NE*	*NE*	
S – pthread_create TH1	10	*pronto*	*esec*	*pronto*	*pronto*	*NE*	*NE*	*NE*	
interrupt da *RT*_clock, scadenza quanto di tempo	20	*pronto*	*pronto*	*esec*	*pronto*	*NE*	*NE*	*NE*	
P – pid1 = fork	30	*pronto*	*pronto*	*esec*	*pronto*	*pronto*	*NE*	*NE*	
P – open	40	*pronto*	*pronto*	*A (open)*	*esec*	*pronto*	*NE*	*NE*	
TH1 – sem_wait	50	*pronto*	*esec*	*A*	*A(s_wait)*	*pronto*	*NE*	*NE*	
S – pthread_create TH2	60	*pronto*	*esec*	*A*	*A*	*pronto*	*pronto*	*NE*	
interrupt da *DMA_in*, tutti i blocchi richiesti trasferiti	70	*pronto*	*pronto*	*esec*	*A*	*pronto*	*pronto*	*NE*	
P – pid2 = fork	80	*pronto*	*pronto*	*esec*	*A*	*pronto*	*pronto*	*pronto*	
P – waitpid (pid1)	90	*pronto*	*pronto*	*A (wait)*	*A*	*esec*	*pronto*	*pronto*	
Q – execl	100	*pronto*	*pronto*	*A (wait)*	*A*	*esec*	*pronto*	*pronto*	
interrupt da *RT*_clock, scadenza quanto di tempo	110	*pronto*	*pronto*	*A (wait)*	*A*	*pronto*	*esec*	*pronto*	
TH2 – mutex_lock	120	*pronto*	*pronto*	*A (wait)*	*A*	*pronto*	*esec*	*pronto*	
TH2 – sem_post	130	*pronto*	*pronto*	*A (wait)*	*esec*	*pronto*	*pronto*	*pronto*	
TH1 – mutex_lock	140	*pronto*	*esec*	*A (wait)*	*A (lock)*	*pronto*	*pronto*	*pronto*	
S – pthread_join (TH2)	150	*pronto*	*A (join)*	*A (wait)*	*A (lock)*	*pronto*	*pronto*	*esec*	
R – write	160	*pronto*	*A (join)*	*A (wait)*	*A (lock)*	*esec*	*pronto*	*A(write)*	
Q – exit	170	*pronto*	*A (join)*	*esec*	*A (lock)*	*NE*	*pronto*	*A(write)*	
P – exit	180	*pronto*	*A (join)*	*NE*	*A (lock)*	*NE*	*esec*	*A(write)*	
TH2 – mutex_unlock	190	*pronto*	*A (join)*	*NE*	*esec*	*NE*	*pronto*	*A(write)*	
TH1 – sem_post	200	*pronto*	*A (join)*	*NE*	*esec*	*NE*	*pronto*	*A(write)*	

esercizio n. 3 – funzioni di *scheduling*

Sono date le condizioni iniziali di uno *scheduling* CFS. Valgono le convenzioni seguenti:

- i tempi di esecuzione sono misurati in *millisecondi* (*ms*)
- i parametri di *CFS* hanno i valori di default: *LT* = 6 *ms*, *GR* = 0,75 *ms* e *WGR* = 1 *ms*

Simulare l'evoluzione del sistema fino al primo evento successivo ai 10 ms

```
CONDIZIONI INIZIALI    ********************************************
RUNQUEUE - NRT ||  PER  || RQL  ||  CURR  || VMIN
            3  || 6,00  || 5,00 ||  t2    || 0,00

TASKS:     ID  ||LOAD   ||  LC   ||   Q    || VRTC ||  SUM   || VRT
  CURRENT  t2  || 3.0   || 0,60  || 3,60  || 0,33 ||  0,00  || 0,00
  RB TREE  t3  || 1.0   || 0,20  || 1,20  || 1,00 ||  1,20  || 1,20
           t1  || 1.0   || 0,20  || 1,20  || 1,00 ||  1,30  || 1,20
Events of task t3:  WAIT at 1.0;   WAKEUP at 4.0;
Events of task t1:  EXIT at 1.0;
```

SOLUZIONE

```
EVENT  ************  TIME    ||    TYPE   || CONTEXT  || RESCHEDULE
                     3,60    ||  Q_SCADE  ||   t2     ||  true

RUNQUEUE - NRT ||  PER  || RQL  ||  CURR  || VMIN
            3  || 6,00  || 5,00 ||  t3    || 1,20

TASKS:     ID  ||LOAD ||  LC   ||   Q    || VRTC ||  SUM   || VRT
  CURRENT  t3  || 1.0 || 0,20  || 1,20  || 1,00 ||  1,20  || 1,20
  RB TREE  t1  || 1.0 || 0,20  || 1,20  || 1,00 ||  1,30  || 1,20
           t2  || 3.0 || 0,60  || 3,60  || 0,33 ||  3,60  || 1,20


EVENT  ************  TIME    ||    TYPE   || CONTEXT  || RESCHEDULE
                     4,60    ||  WAIT     ||   t3     ||  true

RUNQUEUE - NRT ||  PER  || RQL  ||  CURR  || VMIN
            2  || 6,00  || 4,00 ||  t1    || 1,20

TASKS:     ID  ||LOAD ||  LC   ||   Q    || VRTC ||  SUM    || VRT
  CURRENT  t1  || 1.0 || 0,25  || 1,50  || 1,00 ||  1,30  || 1,20
  RB TREE  t2  || 3.0 || 0,75  || 4,50  || 0,33 ||  3,60  || 1,20
_____________
  WAITING  t3  || 1.0 || 0,20  || 1,20  || 1,00 ||  2,20  || 2,20

EVENT  ************  TIME    ||    TYPE   || CONTEXT  || RESCHEDULE
                     5,60    ||  EXIT     ||   t1     ||  true

RUNQUEUE - NRT ||  PER  || RQL  ||  CURR  || VMIN
            1  || 6,00  || 3,00 ||  t2    || 1,20

TASKS:     ID  ||LOAD ||  LC   ||   Q    || VRTC ||  SUM    || VRT
  CURRENT  t2  || 3.0 || 1,00  || 6,00  || 0,33 ||  3,60  || 1,20
  RB VUOTO
_____________
  WAITING  t3  || 1.0 || 0,20  || 1,20  || 1,00 ||  2,20  || 2,20
```

```
EVENT  ************  TIME    ||    TYPE   || CONTEXT  || RESCHEDULE
                    8,60    ||  WAKEUP   ||   t2     ||  false

           tw.vrt+WGR*tw.LC=2,20+1,00*0,25=2,45 < curr.vrt=2,20
RUNQUEUE - NRT ||  PER  || RQL  ||  CURR  || VMIN
            2  || 6,00  || 4,00 ||  t2    || 2,20

TASKS:     ID  ||LOAD ||  LC   ||   Q   || VRTC ||  SUM    || VRT
  CURRENT  t2  || 3.0 || 0,75  || 4,50  || 0,33 ||  6,60  || 2,20
  RB TREE  t3  || 1.0 || 0,25  || 1,50  || 1,00 ||  2,20  || 2,20


EVENT  ************  TIME    ||    TYPE   || CONTEXT  || RESCHEDULE
                    10,10   ||  Q_SCADE  ||   t2     ||  true

RUNQUEUE - NRT ||  PER  || RQL  ||  CURR  || VMIN
            2  || 6,00  || 4,00 ||  t3    || 2,20

TASKS:     ID  ||LOAD ||  LC   ||   Q   || VRTC ||  SUM    || VRT
  CURRENT  t3  || 1.0 || 0,25  || 1,50  || 1,00 ||  2,20   || 2,20
  RB TREE  t2  || 3.0 || 0,75  || 4,50  || 0,33 ||   8,10  || 2,70
```

esercizio n. 4 – memoria e paginazione

esercizio n. 5 – memoria e allocazione

esercizio n. 6 – File System e I/O

GLI ESERCIZI RELATIVI A QUESTI ARGOMENTI SONO STATI OMESSI IN QUANTO NON PIU' SIGNIFICATIVI PER L'ANNO ACCADEMICO 16_17