**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

## SECONDA PARTE di giovedì 4 luglio 2019

**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)** _____

**CON SOLUZIONI (in corsivo)**

## 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):

```
mutex_t mid
sem_t top, bot
int global = 0
```

---

```
void * alpha (void * arg) {
   mutex_lock (&mid)
   sem_wait (&top)
   sem_wait (&bot)
   mutex_unlock (&mid)                          /* statement A */
   global = 1
   sem_post (&bot)                              /* statement B */
   return (void * 2)
} /* end alpha */
```

---

```
void * omega (void * arg) {
   mutex_lock (&mid)
   global = 3                                   /* statement C */
   sem_wait (&top)
   mutex_unlock (&mid)
   sem_post (&top)
   sem_post (&bot)
   sem_wait (&bot)
   return NULL
} /* end omega */
```

---

```
void main ( ) {
   pthread_t th_1, th_2
   sem_init (&top, 0, 1)
   sem_init (&bot, 0, 0)
   create (&th_2, NULL, omega, NULL)
   create (&th_1, NULL, alpha, 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 – alpha** | **th_2 – omega** |
| subito dopo stat. **A** | *ESISTE* | *ESISTE* |
| subito dopo stat. **B** | *ESISTE* | *PUÒ ESISTERE* |
| subito dopo stat. **C** | *PUÒ ESISTERE* | *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** | | |
|---|---|---|---|
| | *mid* | *bot* | *global* |
| subito dopo stat. **A** | *0* | *0* | *3* |
| subito dopo stat. **B** | *0* | *0 / 1* | *1* |
| subito dopo stat. **C** | *1* | *0* | *3* |
| subito dopo stat. **D** | *0* | *0 / 1* | *2* |

**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** | ***wait bot*** | ***lock mid*** | ***0*** |
| **2** | ***wait bot*** | ___ | ***3*** |
| **3** | | | |

## esercizio n. 2 – processi e nucleo

### prima parte – gestione dei processi

```
// programma prova.c
main ( ) {
   pid1 = fork ( )
   if (pid1 == 0) {                  // codice eseguito dal figlio Q
      execl ("/acso/prog_x", "prog_x", NULL)
      exit (-1)
   } else {
      pid1 = wait (&status)
      write (stdout, msg, 25)
   } /* if */
   exit (0)
}  /* prova */
```

```
// programma prog_x.c
pthread_mutex_t ZERO = PTHREAD_MUTEX_INITIALIZER
sem_t RED, BLUE
```

```
void * LESS (void * arg) {
  sem_wait (&BLUE)
  pthread_mutex_lock (&ZERO)
  sem_wait (&BLUE)
  sem_post (&RED)
  pthread_mutex_unlock (&ZERO)
  return NULL
}  /* LESS */
```

```
void * EQUAL (void * arg) {
  pthread_mutex_lock (&ZERO)
  pthread_mutex_unlock (&ZERO)
  sem_wait (&RED)
  return NULL
}  /* EQUAL */
```

```
main ( ) {  // codice eseguito da Q
    pthread_t TH_1, TH_2
    sem_init (&BLUE, 0, 1)
    sem_init (&RED, 0, 0)

    pthread_create (&TH_1, NULL, LESS, NULL)
    pthread_create (&TH_2, NULL, EQUAL, NULL)
    sem_post (&BLUE)
    pthread_join (TH_2, NULL)
    pthread_join (TH_1, NULL)
    exit (1)
}  /* main */
```

```
// programma esempio.c
main ( ) {
    fd = open ("/acso/dati", O_RDWR)
    read (fd, vett, 512)
    exit (1)
```

Un processo ***P*** esegue il programma ***prova*** e crea un figlio ***Q*** che esegue una mutazione di codice (programma ***prog_x***). La mutazione di codice va a buon fine e sono creati i thread ***TH1*** e ***TH2***. Un processo ***S*** esegue il programma ***esempio***.

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** | **S** | **P** | *Q* | *TH1* | *TH_2* | | |
|---|---|---|---|---|---|---|---|---|---|
| | *PID* | **1** | **2** | *3* | *4* | *5* | *6* | | |
| *evento* oppure *processo-chiamata* | *TGID* | **1** | **2** | *3* | *4* | *4* | *4* | | |
| **P –pid1=fork** | **0** | **pronto** | **A read** | **esec** | *pronto* | *NE* | *NE* | | |
| **interrupt da RT_clock e scadenza quanto tempo** | 1 | *pronto* | *A read* | *pronto* | *esec* | *NE* | *NE* | | |
| *Q – execl* | 2 | *pronto* | *A read* | *pronto* | *esec* | *NE* | *NE* | | |
| *interrupt da DMA_in, tutti i blocchi richiesti trasferiti* | 3 | **pronto** | **esec** | **pronto** | **pronto** | **NE** | **NE** | | |
| **interrupt da RT_clock e scadenza quanto tempo** | 4 | *pronto* | *pronto* | *esec* | *pronto* | *NE* | *NE* | | |
| *P: pid1 = wait* | 5 | *pronto* | *pronto* | *A wait* | *esec* | *NE* | *NE* | | |
| *Q – pthread_create TH1* | 6 | *pronto* | *pronto* | *A wait* | *esec* | *pronto* | *NE* | | |
| *Q – pthread_create TH2* | 7 | *pronto* | *pronto* | *A wait* | *esec* | *pronto* | *pronto* | | |
| **interrupt da RT_clock e scadenza quanto tempo** | 8 | *pronto* | *esec* | *A wait* | *pronto* | *pronto* | *pronto* | | |
| *S – exit* | 9 | *pronto* | *non esiste* | *A wait* | *pronto* | *esec* | *pronto* | | |
| *TH1 – sem_wait (blue)* | 10 | *pronto* | *non esiste* | *A wait* | *pronto* | *esec* | *pronto* | | |
| *TH1 – mutex_lock (zero)* | 11 | *pronto* | *non esiste* | *A wait* | *pronto* | *esec* | *pronto* | | |
| *TH1 – sem_wait (blue)* | 12 | *pronto* | *non esiste* | *A wait* | *pronto* | *A s_wait* | *esec* | | |
| *TH2 – mutex_lock (zero)* | 13 | *pronto* | *non esiste* | *A wait* | *esec* | *A s_wait* | *A lock* | | |

**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 | *2* | t1 | 100 | | |
| **TASK** | **ID** | **LOAD** | **LC** | **Q** | **VRTC** | **SUM** | **VRT** |
| **CURRENT** | t1 | 1 | *0.5* | *3* | *1* | 10 | 100.0 |
| **RB** | t2 | 1 | *0.5* | *3* | *1* | 30 | 100.5 |
| | | | | | | | |

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

```
Events of task t1:  WAIT at 0.5;   WAKEUP after 2.5;
Events of task t2:  CLONE at 0.5;  EXIT at 1
```

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** | | |
|---|---|---|---|---|---|---|---|
| | | *0.5* | *WAIT* | *t1* | *true* | | |
| **RUNQUEUE** | **NRT** | **PER** | **RQL** | **CURR** | **VMIN** | | |
| | *1* | *6* | *1* | *t2* | *100.5* | | |
| **TASK** | **ID** | **LOAD** | **LC** | **Q** | **VRTC** | **SUM** | **VRT** |
| **CURRENT** | *t2* | *1* | *1* | *6* | *1* | *30* | *100.5* |
| **RB** | | | | | | | |
| | | | | | | | |
| **WAITING** | *t1* | *1* | | | | *10.5* | *100.5* |
| | | | | | | | |

| **EVENTO** | | **TIME** | **TYPE** | **CONTEXT** | **RESCHED** | | |
|---|---|---|---|---|---|---|---|
| | | *1* | *CLONE* | *t2* | *false* | | |
| **RUNQUEUE** | **NRT** | **PER** | **RQL** | **CURR** | **VMIN** | | |
| | *2* | *6* | *2* | *t2* | *101* | | |
| **TASK** | **ID** | **LOAD** | **LC** | **Q** | **VRTC** | **SUM** | **VRT** |
| **CURRENT** | *t2* | *1* | *0.5* | *3* | *1* | *30.5* | *101* |
| **RB** | *t3* | *1* | *0.5* | *3* | *1* | *0* | *104* |
| | | | | | | | |
| **WAITING** | *t1* | *1* | | | | *10.5* | *100.5* |
| | | | | | | | |

| EVENTO | | TIME | TYPE | CONTEXT | RESCHED | | |
|---|---|---|---|---|---|---|---|
| | | *1.5* | *EXIT* | *t2* | *true* | | |
| **RUNQUEUE** | **NRT** | **PER** | **RQL** | **CURR** | **VMIN** | | |
| | *1* | *6* | *1* | *t3* | *101.5* | | |
| **TASK** | **ID** | **LOAD** | **LC** | **Q** | **VRTC** | **SUM** | **VRT** |
| **CURRENT** | *t3* | *1* | *1* | *6* | *1* | *0* | *104* |
| **RB** | | | | | | | |
| | | | | | | | |
| **WAITING** | *t1* | *1* | | | | *10.5* | *100.5* |
| | | | | | | | |

| EVENTO | | TIME | TYPE | CONTEXT | RESCHED | | |
|---|---|---|---|---|---|---|---|
| | | *3* | *WUP* | *t3* | *true* | | |
| **RUNQUEUE** | **NRT** | **PER** | **RQL** | **CURR** | **VMIN** | | |
| | *2* | *6* | *2* | *t1* | *105.5* | | |
| **TASK** | **ID** | **LOAD** | **LC** | **Q** | **VRTC** | **SUM** | **VRT** |
| **CURRENT** | *t1* | *1* | *0.5* | *3* | *1* | *10.5* | *102.5* |
| **RB** | *t3* | *1* | *0.5* | *3* | *1* | *1.5* | *105.5* |
| | | | | | | | |
| **WAITING** | | | | | | | |
| | | | | | | | |

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

Tempo dell'evento considerato:_____ *3*

Calcolo: _____

*tw.vrt+WGR * tw.LC < curr.vrt?*

*102.5 + 1 * 0.5 = 103 < 105.5 ? → true*

*[ tw.vrt = max (100.5 , 105.5 – 3) = 102.5 ]*

## esercizio n. 3 – memoria e file system

### prima parte – memoria

È dato un sistema di memoria caratterizzato dai seguenti parametri generali (**ATTENZIONE a MAXFREE** ) :

**MAXFREE** = **4** **MINFREE** = **2**

**Situazione iniziale** (esistono due processi P e Q)

**PROCESSO: P** **********

```
 PROCESSO: P  **********
VMA : C  000000400,  2 , R  ,  P  ,  M  , <X, 0>
      S  000000600,  2 , W  ,  P  ,  M  , <X, 2>
      D  000000602,  4 , W  ,  P  ,  A  , <-1,0>
      P  7FFFFFFFC,  3 , W  ,  P  ,  A  , <-1,0>
PT: <c0 :1  R>  <c1 :- ->  <s0 :5  R>  <s1 :- ->   <d0 :s0 W>  <d1 :- ->
    <d2 :3  W>  <d3 :- ->  <p0 :6  W>  <p1 :s2 R>  <p2 :- ->
process P - NPV of PC and SP:  c0, p0
```

**PROCESSO: Q** ********** (**non interessa**) **********

**____MEMORIA FISICA____**(pagine libere: 4)_____

```
00 : <ZP>              ||  01 : Pc0 / Qc0 / <X,0>   ||
02 : Qp0 D             ||  03 : Pd2                 ||
04 : ----              ||  05 : Ps0 / Qs0 / <X,2>   ||
06 : Pp0               ||  07 : ----                ||
08 : ----              ||  09 : ----                ||
```

**____STATO del TLB**_____

```
Pc0 : 01 -  0: 1:   ||   Pp0 : 06 -  1: 0:    ||
Pd2 : 03 -  1: 0:   ||         -----           ||
Ps0 : 05 -  0: 0:   ||         -----           ||
      -----         ||         -----           ||
```

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

**LRU ACTIVE:** PC0

**LRU INACTIVE:** pd2, ps0, pp0, qs0, qp0, qc0

## evento 1: *read* (Ps1), *write* (Pd0)

*la pagina Ps1 viene caricata, SwapIn di Pd0 e scrittura, quindi eliminazione da Swap file*

| **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: |

LRU active: _____ *PD0, PS1, PC0*

LRU inactive: _____*pd2, ps0, pp0, qs0, qpo, qc0*

## evento 2: *write* (Pp1)

*PFRA – 3 pagine da liberare: Qp0, Pp0, Ps0 / Qs0 – carica Qp1 / Pp1 da swap, poi COW di Pp1*

| MEMORIA FISICA | |
|---|---|
| 00: <ZP> | 01: *Pc0 / Qc0 / <X, 0>* |
| 02: *Qp1* | 03: *Pd2* |
| 04: *Ps1/ <X, 3>* | 05: *Pp1* |
| 06: | 07: *Pd0* |
| 08: | 09: |

| SWAP FILE | |
|---|---|
| s0: *Qp0* | s1: *Qd0* |
| s2: *Qp1* | s3: *Pp0* |

LRU active: _____ *PP1, PD0, PS1, PC0*

LRU inactive: _____ *pd2, qc0, qp1*

## evento 3: *read* (Pc0) – 2 *kswapd*

*kswapd invoca PFRA e libera 1 pagina: qp1*

| MEMORIA FISICA | |
|---|---|
| 00: <ZP> | 01: *Pc0 / Qc0 / <X, 0>* |
| 02: | 03: *Pd2* |
| 04: *Ps1/ <X, 3>* | 05: *Pp1* |
| 06: | 07: *Pd0* |
| 08: | 09: |

LRU active: _____ *PC0, pp1, pd0, ps1*

LRU inactive: _____ *pd2, qc0*

**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 : Pc0 / <X, 0>        ||
      02 : Pp0                   ||  03 : ----                ||
      04 : ----                  ||  05 : ----                ||
      06 : ----                  ||  07 : ----                ||
```

Per ciascuno 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, 12000)

| **MEMORIA FISICA** | |
|---|---|
| 00: *<ZP>* | 01: *Pc0 / <X, 0>* |
| 02: *Pp0* | 03: *<F, 0>* |
| 04: *<F, 1>* | 05: *<F, 2>* |
| 06: *----* | 07: *----* |

| | **f_pos** | **f_count** | **numero pagine lette** | **numero pagine scritte** |
|---|---|---|---|---|
| **file F** | *12000* | *1* | *3* | *0* |

## eventi 3-5: *fork* (Q), *lseek* (fd, –10000), *write* (fd, 10)

| MEMORIA FISICA | |
|---|---|
| 00: *<ZP>* | 01: *Pc0 / Qc0 / <X, 0>* |
| 02: *Qp0 D* | 03: *Pp0* |
| 04: *<F, 0>* | 05: *<F, 2>* |
| 06: *----* | 07: *----* |

| | f_pos | f_count | numero pagine lette | numero pagine scritte |
|---|---|---|---|---|
| **file F** | *2010* | *2* | *4* | *0* |

## eventi 6-9: fd1 = *open* (G), *write* (fd1, 7000), *close* (fd), *close* (fd1)

| MEMORIA FISICA | |
|---|---|
| 00: *<ZP>* | 01: *Pc0 / Qc0 / <X, 0>* |
| 02: *Qp0 D* | 03: *Pp0* |
| 04: *<G, 0> D* | 05: *<G, 1> D* |
| 06: *----* | 07: *----* |

| | f_pos | f_count | numero pagine lette | numero pagine scritte |
|---|---|---|---|---|
| **file F** | *2010* | *1* | *4* | *1* |
| **file G** | *---* | *0* | *2* | *2* |

## eventi 10 e 11: *context switch* (Q), *write* (fd, 100)

| | f_pos | f_count | numero pagine lette | numero pagine scritte |
|---|---|---|---|---|
| **file F** | *2110* | *1* | *5* | *1* |

## 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. il numero di pagine di cui può crescere la VMA D, senza dovere modificare la dimensione della TP

| **VMA** del processo **P** | | | | | | | |
|---|---|---|---|---|---|---|---|
| AREA | NPV iniziale | dimensione | R/W | P/S | M/A | nome file | offset |
| C | *0000 0040 0* | *8* | *R* | *P* | *M* | *X* | *0* |
| K | *0000 0060 0* | *4* | *R* | *P* | *M* | *X* | *8* |
| S | *0000 0060 4* | *256* | *W* | *P* | *M* | *X* | *12* |
| D | *0000 0070 4* | *2* | *W* | *P* | *A* | *-1* | *0* |
| M0 | *0000 2000 0* | *2* | *W* | *S* | *M* | *G* | *2* |
| P | *7FFF FFFF C* | *3* | *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 4 | *0* | *0* | *3* | *4* |
| D | 0000 0070 4 | *0* | *0* | *3* | *260* |
| M0 | 0000 2000 0 | *0* | *0* | *256* | *0* |
| P | 7FFF FFFF C | *255* | *511* | *511* | *508* |

2. Numero pagine necessarie

# pag PGD: *1*
# pag PUD: *2*
# pag PMD: *2*
# pag PT: *4*
# pag totali: *9*

3. Numero pagine virtuali occupate dal processo: *275*

4. Rapporto di occupazione: *9 / 275 = 0,033*

5. Numero pagine di crescita della VMA D, senza modifica della dimensione di TP:

   *Con la stessa dimensione di TP la VMA D può crescere di ulteriori 250 pagine virtuali*