



Politecnico di Milano

Dipartimento di Elettronica, Informazione e Bioingegneria

prof.ssa Anna Antola

prof. Luca Breveglieri

prof. Roberto Negrini

prof. Giuseppe Pelagatti

prof.ssa Donatella Sciuto

prof.ssa Cristina Silvano

AXO – Architettura dei Calcolatori e Sistemi Operativi

SECONDA PARTE di 23 luglio 2018

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) _____

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  one,  two
sem_t  port
int global = 0
```

```
void * first (void * arg) {
```

```
    mutex_lock (&one)                                /* statement A */
```

```
    global = 1
```

```
    sem_wait (&port)                                /* statement B */
```

```
    mutex_lock (&two)
```

```
    global = 4
```

```
    mutex_unlock(&one)
```

```
    mutex_unlock (&two)
```

```
    return NULL
```

```
} /* end first */
```

```
void * last (void * arg) {
```

```
    mutex_lock (&two)
```

```
    global = 2
```

```
    mutex_lock (&one)
```

```
    global = 3
```

```
    sem_post (&port)                                /* statement C */
```

```
    mutex_unlock (&one)
```

```
    global = 5
```

```
    mutex_unlock (&two)
```

```
    return NULL
```

```
} /* end last */
```

```
void main ( ) {
```

```
    pthread_t TH_1, TH_2
```

```
    sem_init (& port, 0, 0)
```

```
    create (&TH_2, NULL, last, NULL)
```

```
    create (&TH_1, NULL, first, NULL)
```

```
    join (TH_1, NULL)                                /* 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 | <i>thread</i> | |
|----------------------------|---------------------|--------------------|
| | <i>TH_1 – first</i> | <i>TH_2 – last</i> |
| subito dopo stat. A | 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 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 | |
|----------------------------|-------------------|---------------|
| | <i>port</i> | <i>global</i> |
| subito dopo stat. A | 0 - 1 | 0 - 2 - 3 - 5 |
| subito dopo stat. B | 0 | 1 - 5 |
| subito dopo stat. C | 1 | 3 |
| subito dopo stat. D | 0 | 4 |

Il sistema può andare in stallo (*deadlock*), con uno o più *thread* che si bloccano (con *deadlock* si intende anche un blocco dovuto a un solo *thread* che non potrà mai proseguire), in **un caso**. Si indichi lo statement **dove avviene il blocco**:

| <i>TH_1 – first</i> | <i>TH_2 – last</i> | <i>global</i> |
|---------------------|--------------------|---------------|
| sem_wait(&port) | mutex_lock(&one) | 1 - 2 |

esercizio n. 2 – processi e nucleo

prima parte – gestione dei processi

| | |
|--|--------------------------------|
| // programma prova.c | |
| main () { | |
| fd = open ("/acso/esame", ORDWR) | |
| pidQ = fork () | |
| if (pidQ == 0) { // codice eseguito da Q | |
| read (fd, vett, 2048) // 4 blocchi da trasferire | |
| exit (1) | |
| } else { | |
| execl ("/acso/prog_x", "prog_x", NULL) | |
| exit (-1) | |
| } /* if */ | |
| } /* prova */ | |
| // programma prog_x.c | |
| pthread_mutex_t GATE = PTHREAD_MUTEX_INITIALIZER | |
| sem_t GO | |
| void * FIRST (void * arg) { | void * LAST (void * arg) { |
| (1) pthread_mutex_lock (&GATE) | (3) pthread_mutex_lock (&GATE) |
| sem_post (&GO) | (4) sem_wait (&GO) |
| pthread_mutex_unlock (&GATE) | pthread_mutex_unlock (&GATE) |
| (2) sem_wait (&GO) | (5) sem_wait (&GO) |
| return NULL | sem_post (&GO) |
| } /* FIRST */ | return NULL |
| | } /* LAST */ |
| main () { // codice eseguito da P | |
| pthread_t TH_1, TH_2 | |
| sem_init (&GO, 0, 1) | |
| pthread_create (&TH_2, NULL, LAST, NULL) | |
| pthread_create (&TH_1, NULL, FIRST, NULL) | |
| exit (1) | |
| } /* main */ | |

Un processo **P** esegue il programma **prova** e crea il processo **Q**. **P** esegue quindi una mutazione di codice che va **a buon fine**. Nel codice mutato **P** crea i thread **TH1** e **TH2**.

Si simuli l'esecuzione dei processi completando tutte le righe presenti nella tabella così come risulta dal codice dato, dallo stato iniziale, dagli eventi indicati e ipotizzando che il processo **P** **non abbia ancora eseguito la execl**. Si completi la tabella riportando quanto segue:

- $\langle PID, TGID \rangle$ di ciascun processo che viene creato
- $\langle \text{evento oppure identificativo del processo-chiamata di sistema / libreria} \rangle$ nella prima colonna, dove necessario e in funzione del codice proposto (le istruzioni da considerare sono evidenziate in grassetto)
- 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)

| <i>identificativo simbolico del processo</i> | | IDLE | P | Q | TH2 | TH1 | | |
|--|-------------|---------------|-------------|---------------|------------|---------------|--|--|
| <i>evento/processo-chiamata</i> | <i>PID</i> | 1 | 2 | 3 | 4 | 5 | | |
| | <i>TGID</i> | 1 | 2 | 3 | 2 | 2 | | |
| Q – read | 0 | pronto | ESEC | A read | NE | NE | | |
| 1 interrupt da DMA_in, non è l'ultimo blocco | 10 | pronto | ESEC | A read | NE | NE | | |
| P - execl | 20 | pronto | ESEC | A read | NE | NE | | |
| P - pthread_create(TH2) | 30 | pronto | ESEC | A read | pronto | NE | | |
| Interrupt da DMA_IN, letti tutti i blocchi (3) | 40 | pronto | pronto | exec | pronto | non esiste | | |
| interrupt da RT_clock, scadenza quanto di tempo | 50 | pronto | pronto | pronto | ESEC | NE | | |
| TH2 - mutex_lock(&LAST) | 60 | pronto | pronto | pronto | ESEC | NE | | |
| TH2 - sem_wait(&GO) | 70 | pronto | pronto | pronto | ESEC | NE | | |
| interrupt da RT_clock, scadenza quanto di tempo | 80 | pronto | ESEC | pronto | pronto | NE | | |
| P - pthread_create(TH1) | 90 | pronto | ESEC | pronto | pronto | pronto | | |
| interrupt da RT_clock, scadenza quanto di tempo | 100 | pronto | pronto | ESEC | pronto | pronto | | |
| Q - exit | 110 | pronto | pronto | NE | ESEC | pronto | | |

seconda parte – struttura e moduli del nucleo

Riferimento: funzione di libreria NPTL **sem_wait** e sua implementazione tramite **futex**.

Evento: un processo Q invoca *sem_wait* e la sua esecuzione comporta l'invocazione di *system_call*.

Indicare l'invocazione di tutti i moduli di libreria e eventualmente di SO per la gestione dell'evento che si svolgono nel contesto del processo Q. Mostrare il relativo contenuto delle pile utente e di sistema.

NOTAZIONE da usare per i moduli: > (invocazione), nome_modulo (esecuzione), < (ritorno)

IL NUMERO DI RIGHE VUOTE NELLE TABELLE QUI SOTTO NON È SIGNIFICATIVO

| processo | modo | modulo |
|----------|--------|--------------------------------------|
| Q | U | > sem_wait |
| Q | U | > syscall |
| Q | U => S | SYSCALL |
| Q | S | > system_call |
| Q | S | > sys_futex (wait) |
| Q | S | > wait_event_interruptible_exclusive |
| Q | S | > schedule |
| Q | S | > pick_next_task < |
| Q' | S | context-switch |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

| | |
|---------|---|
| | |
| | |
| | |
| | Indirizzo di rientro a syscall da sem_wait |
| | Indirizzo di rientro al codice utente da sem_wait |
| W | piena |
| | piena |
| ubase_Q | piena |

uStack_Q

Pop dalla pila =>

| | |
|---------|---|
| | |
| | |
| | |
| | Indirizzo di rientro a schedule da pick_next_task |
| | Indirizzo di rientro a wait_event da schedule |
| | Indirizzo di rientro a sys_futex da wait_event |
| | Indirizzo di rientro a system_call da sys_futex |
| | PSR U |
| sbase_Q | Indirizzo di rientro a syscall da system_call |

sStack_Q

esercizio n.3 – gestione della memoria e file system

prima parte – gestione della memoria

È dato un sistema di memoria caratterizzato dai seguenti parametri generali: **MAXFREE = 4**, **MINFREE = 3**.
Si consideri la seguente **situazione iniziale**

PROCESSO: P ...

PROCESSO: Q ...

| MEMORIA FISICA (pagine libere: 3) | | | |
|-----------------------------------|--|--------------------|--|
| 00 : <ZP> | | 01 : Pc1/Qc1/<X,1> | |
| 02 : Pp0/Qp0 | | 03 : <X,2> | |
| 04 : Ps0/Qs0 | | 05 : Qp1 D | |
| 06 : Pp1 | | 07 : ---- | |
| 08 : ---- | | 09 : ---- | |

| STATO del TLB | | | |
|------------------|--|------------------|--|
| Pc1 : 01 - 0: 1: | | Pp0 : 02 - 1: 0: | |
| Ps0 : 04 - 1: 0: | | Pp1 : 06 - 1: 1: | |
| ---- | | ---- | |

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

LRU ACTIVE: QP1, QC1, PP1, PC1,

LRU INACTIVE: qs0, qp0, ps0, pp0,

Si rappresenti l'effetto dei seguenti eventi consecutivi sulle strutture dati della memoria compilando esclusivamente le tabelle fornite per ciascun evento (l'assenza di una tabella significa che non è richiesta la compilazione della corrispondente struttura dati).

ATTENZIONE: le Tabelle sono PARZIALI – riempire solamente le celle indicate

evento 1: *write* (Ps0)

| MEMORIA FISICA | |
|----------------|------------------------|
| 00: <ZP> | 01: Pc1 / Qc1 / <X, 1> |
| 02: Ps0 | 03: ---- |
| 04: Qs0 (D) | 05: Qp1 (D) |
| 06: Pp1 | 07: ---- |
| 08: ---- | 09: ---- |

| TLB | | | | | | | |
|-----|-----|---|---|-----|-----|---|---|
| NPV | NPF | D | A | NPV | NPF | D | A |
| Pc1 | 1 | 0 | 1 | Ps0 | 2 | 1 | 1 |
| Pp1 | 6 | 1 | 1 | | | | |

| SWAP FILE | |
|---------------|-----|
| s0: Pp0 / Qp0 | s1: |
| s2: | s3: |

LRU INACTIVE: ps0, qs0

evento 2: *mmap* (0x 000050000000, 3, W, P, M, "F", 2),

| VMA del processo P (compilare solo la riga relativa alla nuova VMA creata) | | | | | | | |
|---|----------------|------------|-----|-----|-----|-----------|--------|
| AREA | NPV iniziale | dimensione | R/W | P/S | M/A | nome file | offset |
| M0 | 0x 0000 5000 0 | 3 | W | P | M | F | 2 |

| PT del processo: P | | | | |
|--------------------|-------------|-------------|-------------|-------------|
| s0: <:2 W> | p0: <:s0 R> | m00: <:- -> | m01: <:- -> | m02: <:- -> |

evento 3: read (Pm01, Pm02)

| PT del processo: P | | | | |
|--------------------|-------------|-------------|-------------|-------------|
| s0: <:s1 W> | p0: <:s0 R> | m00: <:- -> | m01: <:3 R> | m02: <:2 R> |

| MEMORIA FISICA | |
|-------------------|------------------------|
| 00: <ZP> | 01: Pc1 / Qc1 / <X, 1> |
| 02: Pm02 / <F, 4> | 03: Pm01 / <F, 3> |
| 04: ---- | 05: Qp1 (D) |
| 06: Pp1 | 07: ---- |
| 08: ---- | 09: ---- |

| SWAP FILE | |
|---------------|---------|
| s0: Pp0 / Qp0 | s1: Ps0 |
| s2: Qs0 | s3: |

seconda parte – memoria e file system

È dato un sistema di memoria caratterizzato dai seguenti parametri generali:

MAXFREE = 2 MINFREE = 1

Si consideri la seguente **situazione iniziale**:

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

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 sempre in esecuzione il processo **P**.

ATTENZIONE: il numero di pagine lette o scritte è cumulativo, quindi è la somma delle pagine lette o scritte da tutti gli eventi precedenti oltre a quello considerato.

evento 1 – *fd = open (F)*

| f_pos | f_count | numero pagine lette | numero pagine scritte |
|-------|---------|---------------------|-----------------------|
| 0 | 1 | | |

evento 2 – *read (fd, 5500)*

| MEMORIA FISICA | |
|----------------|------------------------|
| 00: <ZP> | 01: Pc0 / Qc0 / <X, 0> |
| 02: Pp0 / Qp0 | 03: Qp1 (D) |
| 04: Pp1 | 05: <F, 0> |
| 06: <F, 1> | 07: |

| f_pos | f_count | numero pagine lette | numero pagine scritte |
|-------|---------|---------------------|-----------------------|
| 5500 | 1 | 2 | 0 |

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

evento 3 – *write* (fd, 1000)

$$5500 + 1000 = 6500$$

| MEMORIA FISICA | |
|----------------|------------------------|
| 00: <ZP> | 01: Pc0 / Qc0 / <X, 0> |
| 02: Pp0 / Qp0 | 03: Qp1 (D) |
| 04: Pp1 | 05: <F, 0> |
| 06: <F, 1> (D) | 07: |

| f_pos | f_count | numero pagine lette | numero pagine scritte |
|-------|---------|---------------------|-----------------------|
| 6500 | 1 | 2 | 0 |

evento 4 – *write* (fd, 5500)

$$6500 + 5500 = 12000$$

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

Prima faccio READ & WRITE su <F, 1>. Poi devo fare READ & WRITE su <F, 2>, ma non ho spazio, quindi, PFRA

| f_pos | f_count | numero pagine lette | numero pagine scritte |
|-------|---------|---------------------|-----------------------|
| 12000 | 1 | 3 | 1 |

evento 5 – *close* (fd)

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

| f_pos | f_count | numero pagine lette | numero pagine scritte |
|-------|---------|---------------------|-----------------------|
| | | 3 | 2 |

esercizio n. 4 – Domanda - Struttura 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 | 3 | R | P | M | X | 0 |
| K | 0000 0060 0 | 1 | R | P | M | X | 3 |
| S | 0000 0060 1 | 256 | W | P | M | X | 4 |
| D | 0000 0070 1 | 2 | W | P | A | -1 | 0 |
| M0 | 0000 1000 0 | 1 | W | S | M | G | 2 |
| M1 | 0000 3000 0 | 3 | W | P | M | F | 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 1 | 0 | 0 | 3 | 1 |
| D | 0000 0070 1 | 0 | 0 | 3 | 257 |
| M0 | 0000 1000 0 | 0 | 0 | 128 | 0 |
| M1 | 0000 3000 0 | 0 | 0 | 384 | 0 |
| P | 7FFF FFFF C | 255 | 511 | 511 | 508 |

2. Numero pagine necessarie

pag PGD: 1

pag PUD: 2

pag PMD: 2

pag PT: 5

pag totali: 10

3. Numero pagine virtuali occupate dal processo: 269

4. Rapporto di occupazione: $10 / 269 = 3.72\%$

5. Dimensione massima del processo in pagine virtuali: $509 + 509 + 511 + 253 + 509 + 269 = 2560$

6. Rapporto di occupazione con dimensione massima: $10 / 2560 = 0.39\%$

C => 0x 0000 0040 0 => 0000 0000 0000 0000 0000 0000 0100 0000 0000 => 0 0 2 0

K => 0x 0000 0060 0 => 0000 0000 0000 0000 0000 0000 0110 0000 0000 => 0 0 3 0

S => 0x 0000 0060 1 => 0000 0000 0000 0000 0000 0000 0110 0000 0001 => 0 0 3 1

D => 0x 0000 0070 1 => 0000 0000 0000 0000 0000 0000 0111 0000 0001 => 0 0 3 257

M0 => 0x 0000 1000 0 => 0000 0000 0000 0000 0001 0000 0000 0000 0000 => 0 0 128 0

M1 => 0x 0000 3000 0 => 0000 0000 0000 0000 0011 0000 0000 0000 0000 => 0 0 384 0

P => 7FFF FFFF C => 0111 1111 1111 1111 1111 1111 1111 1111 1100 => 255 511 511