Tutoraggio di Sistemi Operativi

Comunicazione tra Processi e Thread

Matteo Federico

Lezione #9

TOR VERGATA

UNIVERSITÀ DEGLI STUDI DI ROMA

Esercizio 4 - Esame 13/6/2022

Si consideri un insieme di N processi (P1 , . . . , PN) ed un altro insieme di M processi (L1 , . . . , LM).
Ogni processo Pi scrive periodicamente un nuovo messaggio in una memoria condivisa M, che deve essere letto una sola volta da tutti i processi Lj.
Il processo Pi può scrivere un nuovo messaggio solo dopo che l'ultimo messaggio scritto sia stato letto da tutti i processi Li.
Altrimenti dovrà rimanere in attesa. Allo stesso tempo, un processo Pi che ha scritto un messaggo in M ne può scrivere un successivo solo dopo che anche tutti gli altri processi Pj (con j diverso da i) abbiano scritto il loro messaggio in M. Altrimenti dovrà rimanere in attesa. Allo stesso tempo, ogni processo Lj che intende leggere dovrà rimanere in attesa che un messaggio non ancora letto d Lj sia reso disponibile.
Si schematizzi la soluzione del suddetto problema di sincronizzazione, usando solo semafori, fornendo lo pseudo-codice delle procedure SCRIVI e LEGGI usate rispettivamente dai generici processi Pi e Lj.

TOR VERGATA

13/6/2022 — N scrittori, M lettori (barriera + broadcast)

Problema

- N processi P_i scrivono su memoria condivisa m
- Ogni messaggio letto da *tutti* gli M processi L_j (esattamente una volta)
- P_i riscrive solo dopo che tutti L_j abbiano letto
- P_i deve aspettare che tutti gli altri P_j abbiano scritto (barriera/Ordine)

Strutture dati

```
1 //N+M semafori binari
2 sem_t s_writer[N];        // init={1,0,...,0} Uno solo sbloccato
3 sem_t ready_reader[M];       // init={0,...,0}
4 //un semaforo da 0 a M
5 sem_t s_reader_finish;    // init=0
```

13/6/2022 — Pseudo-codice SCRIVI e LEGGI

SCRIVI – processo P_i

```
1  procedure SCRIVI(i, msg):
2      wait(s_writer[i],1)
3      M = msg
4      // sblocca tutti i lettori
5      for j in 1..M:
6          signal(ready_read[j],1)
7      // barriera lettori
8      wait(s_reader_finish,M)
9      signal(s_writer[(i+1) mod N])
```

LEGGI – processo L_j

```
1  procedure LEGGI(j):
2      wait(ready_read[j],1) //Attendo lo scrittore
3      msg = M       //leggo i dati
4      signal(s_reader_finish,1) //segnalo che ho
         letto il buffer
5      return msg
```

TOR VERGATA

Esercizio 4 - Esame 18/7/2022

Si consideri un sistema con due processi PROC1 e PROC2 che scambiano periodicamente informazioni utilizzando due segmenti di memoria condivisa M1 e M2.
Lo scambio delle informazioni avviene secondo il seguente schema: PROC1 scrive un nuovo messaggio in M1 mentre PROC2 scrive un nuovo messaggio in M2, PROC2 scrive una risposta per un messaggio di PROC1 in M1 mentre PROC1 scrive una risposta per un messaggio di PROC2 in M2.
Quando qualsiasi dei due processi (PROC1 o PROC2) intende scambiare informazioni scrivendo il proprio messaggio nel relativo slot della memoria condivisa, esso deve rimanere in attesa che anche l'altro processo intenda scambiare informazioni scrivendo il suo messaggio.
In particolare, dopo aver scritto un nuovo messaggio, ogni processo deve rimanere in attesa della risposta da parte dell'altro processo. Si schematizzi la soluzione del suddetto problema di sincronizzazione, usando solo semafori, fornendo lo pseudo-codice delle procedure per lo scambio di informazioni usate da PROC1 e PROC2.

TOR VERGATA

18/7/2022— Due processi, due memorie, scambio sincronizzato

Problema

- PROC1 e PROC2 si scambiano messaggi su M_1 e M_2
- Scrivono *contemporaneamente*: PROC1$\rightarrow M_1$, PROC2$\rightarrow M_2$ (rendez-vous)
- Dopo aver scritto, ciascuno attende la risposta dell'altro. (l'uno scrive nel buffer dell'altro)

Strutture dati

```
1 sem_t wait_proc1;     // init = 0 indica che PROC1 e' pronto
2 sem_t wait_proc2;   // initi = 0 indica che PROC2 e' pronto
3 sem_t ready_M1;   // init=0 indica che in M1 ci sono dati
4 sem_t ready_M2;   // init=0 indica che in M2 ci sono dati
```

18/7/2022 — Pseudo-codice PROC1 e PROC2

PROC1

```
1  procedure SCAMBIO_PROC1(msg):
2       //segnalo che sono pronto
3      signal(wait_proc1,1)
4      //attendo l'altro
5      wait(wait_proc2,1)
6      M1 = msg
7      signal(ready_M1,1)
8      wait(ready_M2,1)
9      msg = M2
10     M2 = REPLAY(msg)
11     //segnalo il replay pronto
12     signal(ready_M2,1)
13     //attendo il replay di PROC2
14     wait(ready_M1,1)
15     msg = M1
16     return msg
```

PROC2 (simmetrica)

```
1  procedure SCAMBIO_PROC2(msg):
2       //segnalo che sono pronto
3      signal(wait_proc2,1)
4      //attendo l'altro
5      wait(wait_proc1,1)
6      M2 = msg
7      signal(ready_M2,1)
8      wait(ready_M1,1)
9      msg = M1
10     M1 = REPLAY(msg)
11     //segnalo il replay pronto
12     signal(ready_M1,1)
13     //attendo il replay di PROC1
14     wait(ready_M2,1)
15     msg = M2
16     return msg
```

TOR VERGATA

Esercizio 4 - Esame 8/9/2022

Si consideri un insieme di N processi P1, P2, P3, ...,PN ed un ulteriore processo PROD. Il processo PROD periodicamente attende che almeno N/2 processi Pi generici siano pronti a leggere un nuovo messaggio, e deposita tale nuovo messaggio su una memoria condivisa M, abilitando la lettura esattamente per N/2 processi Pi.
Ogni processo Pi periodicamente vorrá acquisire un nuovo messaggio di PROD, e rimarrá in attesa fino a che non sia riuscito a leggere tale nuovo messaggio.
Si schematizzi la soluzione del suddetto problema di sincronizzazione, usando solo semafori, fornendo lo pseudo-codice delle procedure LEGGI usata dai processi Pi e SCRIVI usata dal processo PROD

TOR VERGATA

8/9/2022 — PROD scrive per $N/2$ lettori - Version1

Problema

- PROD aspetta che almeno $N/2$ processi P_i siano pronti
- Deposita un messaggio su m e abilita *esattamente* $N/2$ letture
- Ogni P_i si registra e poi attende il permesso di leggere
- Ogni messaggio viene letto solo da N/2 lettori

Strutture dati

```
1 sem_t reader;        // init=0  (valore massimo N)
2 sem_t ready;         // init=0  (valore massimo N/2)
```

8/9/2022 — Pseudo-codice LEGGI (P_i) e SCRIVI (PROD)

LEGGI – processo P_i

```
1 procedure LEGGI(i):
2   // registrazione
3   signal(reader,1)
4   wait(ready,1)
5   val = M
```

SCRIVI – PROD

```
1 procedure SCRIVI(msg):
2   // attendi N/2 Pi pronti
3   wait(reader,N/2)
4   M = msg
5   signal(ready,N/2)
```

8/9/2022 — PROD scrive per $N/2$ lettori - Versione 2

Problema

- PROD aspetta che almeno $N/2$ processi P_i siano pronti
- Deposita un messaggio su m e abilita *esattamente* $N/2$ letture
- Ogni P_i si registra e poi attende il permesso di leggere
- Ogni messaggio viene letto da tutti i processi ma solo N/2 lettori alla volta

Strutture dati

```
1 sem_t reader;        // init=0   (valore massimo N)
2 sem_t ready;         // init=0   (valore massimo N/2)
3 sem_t reader_f;      // init=0   (valore massimo N/2)
4 sem_t reader_all;    // init=0   (valore massimo N)
```

TOR VERGATA

8/9/2022 — Pseudo-codice LEGGI (P_i) e SCRIVI (PROD)

LEGGI – processo P_i

```
1  procedure LEGGI(i):
2    // registrazione
3    signal(reader,1)
4    wait(ready,1)
5    val = M
6    signal(reader_f,1)
7    wait(reader_all,1)
```

SCRIVI – PROD

```
1  procedure SCRIVI(msg):
2    // attendi N/2 Pi pronti
3    wait(reader,N/2)
4    M = msg
5    signal(ready,N/2)
6    //aspetto che chi ha letto finisca
7    wait(reader_f,N/2)
8    //aspetto che ce ne sono altri N/2
9    wait(reader,N/2)
10   signal(ready,N/2)
11   wait(reader_f,N/2)
12   signal(reader_all,1)
```

TOR VERGATA

Esercizio 4 - Esame 21/6/2021

Si consideri un insieme di N processi P0, P1, P2, P3, ..., PN-1, e una memoria condivisa M composta da N/2 slot. Il processo P[i] periodicamente attende che un nuovo messaggio venga depositato su M[N/2 – (i mod(N/2))]. Un ulteriore processo PROD scrive periodicamente messaggi su M secondo uno schema buffer circolare. Ogni messaggio scritto da PROD in M[i] può essere letto da un solo processo dell'insieme P0, P1, P2, P3, ..., PN-1. Inoltre, ogni processo Pi non può leggere entrambi i messaggi di una coppia di messaggi consecutivi depositati da PROD in uno stesso slot di M.
Se all'atto della lettura Pi non ha la possibilità di leggere il messaggio correntemente nello slot oppure un messaggio da lui leggibile non sia presente, Pi deve entrare in stato di attesa. Inoltre, PROD deve entrare in stato di attesa se all'atto della scrittura il corrispettivo slot di M contenga un messaggio non ancora letto.
Si schematizzi la soluzione del suddetto problema di sincronizzazione, usando solo semafori, fornendo lo pseudo-codice delle procedure SCRIVI e LEGGI usate, rispettivamente, da PROD e da ciascuno dei processi Pi.

TOR VERGATA

21/6/2021 — Buffer circolare $N/2$ slot, N processi

Problema

- Considero solo il caso pari
- $\mathcal{M}$ ha $N/2$ slot; P_i legge $\mathcal{M}[\, N/2 - 1 - (i \bmod N/2)\,]$
- PROD scrive circolarmente: slot $0, 1, \ldots, N/2-1, 0, \ldots$
- Ogni msg letto da *un solo* P_i
- P_i non può leggere due messaggi consecutivi dallo stesso slot
- PROD attende se lo slot contiene un msg non ancora letto

Strutture dati

```
1 sem_t s_write[N/2];          // init={1,..,1}
2 sem_t s_write_read[N/2];     // init={0,...,0}
3 sem_t s_read_read[N];        // init={1,..,1,0,..,0} i primi N/2 1 e i secondi N/2 0
```

21/6/2021 — Pseudo-codice SCRIVI (PROD) e LEGGI (P_i)

SCRIVI – PROD

```
1 procedure SCRIVI(msg):
2     for i in (0,N/2):
3         wait(s_write[i],1)
4         M[i] = msg[i]
5         signal(s_write_read[i],1)
```

LEGGI – processo P_i

```
1 procedure LEGGI(i):
2     j=N/2 - 1 - (i mod(N/2))
3     wait(s_read_read[i],1)
4     wait(s_write_read[j],1)
5     msg = M[j]
6     signal(s_write[j],1)
7     //Trovo il lettore a me gemello
8     K = (i+N/2) mod N
9     signal(s_read_read[k],1)
```

TOR VERGATA

Esercizio 4 - Esame 19/7/2021

Si considerino due insiemi di processi A = P0, P1, P2 ... Pn e B = R0, R1, R2 .. Rn, e una memoria condivisa M composta da un solo slot. I processi nell'insieme A scrivono periodicamente un nuovo messaggio su M. I processi nell'insieme B leggono il messaggio e inseriscono sempre su M un messaggio di risposta. La regola di lettura è tale per cui se il messaggio su M è scritto da Pi allora la lettura dovrà essere effettuata da Ri, e la risposta di Ri dovrà essere letta da Pi prima che un nuovo messaggio possa essere depositato su M. D'altro canto, quando uno qualsiasi dei processi deve leggere un messaggio o una risposta e questo non è presente su M allora il processo deve entrare in attesa. Inoltre ogni messaggio o risposta scritto su M non può essere sovrascritto prima di essere letto e non può essere letto più di una volta. Si schematizzi la soluzione del suddetto problema di sincronizzazione, usando solo semafori, fornendo lo pseudo-codice delle procedure SCRIVI e RISPONDI usate, rispettivamente, da ciascuno dei processi negli insiemi A e B

TOR VERGATA

19/7/2021 — A scrive, R_i risponde, P_i legge risposta

Problema

- $A = \{P_0, \ldots, P_n\}$ scrivono su $\mathcal{M}$; $B = \{R_0, \ldots, R_n\}$ rispondono
- Se P_i scrive, *solo* R_i legge e risponde
- P_i legge la risposta prima che un nuovo msg sia depositato
- Nessuna sovrascrittura né doppia lettura

Strutture dati

```
1 sem_t slot_free;          // init=1
2 sem_t msg_ready[N+1];     // init=0
3 sem_t reply_ready[N+1];   // init=0
```

Ciclo di vita di un messaggio

19/7/2021 — Pseudo-codice SCRIVI (P_i) e RISPONDI (R_i)

SCRIVI – processo P_i

```
1  procedure SCRIVI(i, msg):
2    // acquisisce lo slot
3    wait(slot_free)
4    M = msg
5    // notifica il proprio Ri
6    signal(msg_ready[i])
7    // attende risposta
8    wait(reply_ready[i])
9    risposta = M
10   // libera lo slot
11   signal(slot_free)
12   return risposta
```

RISPONDI – processo R_i

```
1  procedure RISPONDI(i):
2    // attende messaggio di Pi
3    wait(msg_ready[i])
4    msg = M
5    M = elabora(msg)
6    signal(reply_ready[i])
```

TOR VERGATA

Esercizio 4 - Esame 6/9/2021

Si consideri un insieme di 8 processi P0, P2, P3, P4, . . . , P6, P7 ciascuno dei quali scrive periodicamente un nuovo messaggio su uno slot di una memoria condivisa M a 4 slot. Il processo Pi scrive esclusivamente sul corrispettivo slot M[i] della memoria condivisa, con 'i' pari all'indice del processo modulo 4. Un ulteriore processo REPLY legge periodicamente in ordine circolare i messaggi scritti dai processi Pi. Ogni 2 nuovi messaggi letti, REPLY scrive una risposta su una memoria condivisa R, e la risposta deve essere letta dai 2 processi Pi mittenti dei messaggi letti da REPLY. La lettura di REPLY dagli slot della memoria condivisa M è bloccante in assenza di nuovi messaggi, così come la lettura della risposta da parte dei processi Pi su R in assenza di risposta, mentre le scritture sugli slot di M sono concorrenti. Si schematizzi la soluzione del suddetto problema di sincronizzazione, usando solo semafori, fornendo lo pseudo-codice delle procedure SCRIVI-LEGGI usata dai processi Pi e LEGGI-RISPONDI usata dal processo REPLY.

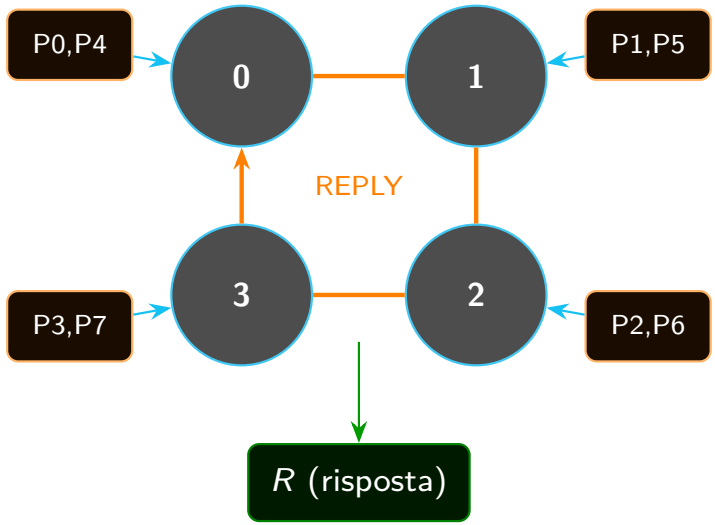
TOR VERGATA

6/9/2021 — 8 processi, 4 slot, REPLY legge 2 msg e risponde

Problema

- 8 processi P_i scrivono su $\mathcal{M}[i \bmod 4]$ (scritture concorrenti)
- REPLY legge circolarmente slot $0, 1, 2, 3, 0, \ldots$
- Ogni **2 messaggi letti**, REPLY scrive una risposta su R
- La risposta va letta dai 2 P_i mittenti

Schema circolare



ogni 2 msg $\longrightarrow$ signal(reply_sem[s0,s1])

Strutture dati

```
1 sem_t slot_full[4];   // init=0
2 sem_t slot_free[4];   // init=1
3 sem_t reply_sem[8];   // init=0
4 int M[4]; int R;
5 int sender[2]; // mittenti batch
6 int reply_idx = 0;
```

6/9/2021 — Pseudo-codice SCRIVI-LEGGI (P_i) e LEGGI-RISPONDI (REPLY)

SCRIVI-LEGGI – P_i

```
1  procedure SCRIVI_LEGGI(i, msg):
2    k = i mod 4
3    // attendi slot libero
4    wait(slot_free[k])
5    // scrivi con identificativo
6    M[k] = (i, msg)
7    signal(slot_full[k])
8    // attendi risposta
9    wait(reply_sem[i])
10   risposta = R
11   return risposta
```

LEGGI-RISPONDI – REPLY

```
1  procedure LEGGI_RISPONDI():
2    loop:
3      // leggi 1o messaggio
4      wait(slot_full[reply_idx])
5      (s0,m0) = M[reply_idx]
6      signal(slot_free[reply_idx])
7      reply_idx=(reply_idx+1) mod 4
8      // leggi 2o messaggio
9      wait(slot_full[reply_idx])
10     (s1,m1) = M[reply_idx]
11     signal(slot_free[reply_idx])
12     reply_idx=(reply_idx+1) mod 4
13     // risposta per entrambi
14     R = elabora(m0, m1)
15     signal(reply_sem[s0])
16     signal(reply_sem[s1])
```