Concorrenza (Parte 2)

Soluzioni software per la sincronizzazione

Corso: Sistemi di Calcolo 2
Docente: Riccardo Lazzeretti

Nota sulle fonti

Queste note sono una traduzione fedele e una riorganizzazione del materiale delle slide *"Concurrency again / Software solutions for synchronization"*. Dove le slide contenevano figure/diagrammi temporali, qui offriamo una descrizione testuale precisa.

Indice

1 Algoritmo di Dijkstra per la mutua esclusione (N processi)

1.1 Pseudocodice (dalle slide)

Listing 1: Algoritmo di Dijkstra (versione presentata nelle slide)

```
/* Memoria globale */
boolean interested[N] = {false, ..., false};
boolean passed[N]     = {false, ..., false};
int k = /* un qualsiasi valore in {0, 1, ..., N-1} */;

/* Dati locali del processo i (i in {0, ..., N-1}) */
while (true) {
  interested[i] = true;
  while (k != i) {
    passed[i] = false;
    if (!interested[k]) k = i;
  }
  passed[i] = true;
  for (int j = 0; j < N; ++j) if (j != i) {
    if (passed[j]) goto retry;   // conflitto => riprova
  }
  /* ---- Sezione critica ---- */

  passed[i] = false;
  interested[i] = false;
  continue;
retry:
  ; // ritorna al while(k != i)
}
```

1.2 Caratteristiche riportate

- **Mutua esclusione**: garantita.
- **Assenza di deadlock**: garantita.
- **Assenza di starvation**: *non garantita.*
- **Requisiti**: lettura/scrittura atomiche delle variabili condivise; memoria condivisa per la variabile k.

2 Algoritmo *Bakery* di Lamport (1975)

2.1 Idea intuitiva ("prendi il numeretto")

Come in un negozio affollato: ciascun cliente prende un biglietto con un numero; quando più clienti sono in attesa, l'ordine di servizio è determinato dall'ordine dei numeri (a parità di numero, prevale l'ID più piccolo). Se nessuno è in attesa, i biglietti non contano.

2.2 Struttura concettuale del flusso

Descrizione testuale dei diagrammi temporali nelle slide: i processi attraversano tre regioni logiche — *remainder* (fuori dalla ME), *doorway* (scelta del numero: fase brevissima che deve apparire *quasi atomica*) e *entry/waiting* (attesa ordinata), poi *CS* (sezione critica) e *exit*. La corretta gestione della *doorway* è fondamentale per evitare anomalie.

2.3 Implementazione 1 (errata): solo numeri

Listing 2: Implementazione 1 (dalle slide) — NON corretta

```
/* per il processo i, i in {1..N} */
while (true) {
  /* NCS */
  number[i] = 1 + max{ number[j] | 1 <= j <= N, j != i };
  for (int j = 1; j <= N; ++j) if (j != i) {
    while (number[j] != 0 && number[j] < number[i]) ;  // attesa
  }
  /* CS */
  number[i] = 0;
}
```

Esito delle slide **Non** soddisfa la mutua esclusione e può andare in **deadlock**. I diagrammi mostrano scenari in cui due processi si bloccano o entrano insieme in CS a causa di letture non coordinate del massimo.

2.4 Variante con $\leq$ (ancora errata)

Sostituire $<$ con $\leq$ non risolve: persiste la possibilità di deadlock (come illustrato nelle sequenze temporali).

2.5 Implementazione 2 (errata): ordine lessicografico senza `choosing`

Listing 3: Implementazione 2 (dalle slide) — NON corretta

```
while (true) {
  /* NCS */
  number[i] = 1 + max{ number[j] | j != i };
  for (int j = 1; j <= N; ++j) if (j != i) {
    // ordine lessicografico: (B,j) < (A,i) sse (B < A) || (B == A && j < i)
    while (number[j] != 0 && (number[j], j) < (number[i], i)) ;
  }
  /* CS */
  number[i] = 0;
}
```

Esito delle slide Anche così, **non** si garantisce la mutua esclusione: i diagrammi mostrano interleaving che portano due processi in CS insieme.

2.6 Algoritmo *Bakery* corretto

La correzione introduce un vettore di bit `choosing[]` per segnalare che un processo sta scegliendo il proprio numero, garantendo l'effetto "porta stretta" (*doorway*) e rimuovendo le gare durante la lettura del massimo e il confronto lessicografico.

Listing 4: Algoritmo *Bakery* di Lamport (corretto)

```
/* Variabili globali */
volatile bool choosing[N] = { false, ... };
volatile int  number[N]   = { 0, ... };

/* Codice del processo i (i in {1..N}) */
while (true) {
  /* NCS */

  /* doorway */
  choosing[i] = true;
  number[i] = 1 + max{ number[j] | 1 <= j <= N, j != i };
  choosing[i] = false;

  /* entry/waiting */
  for (int j = 1; j <= N; ++j) if (j != i) {
    while (choosing[j]) ;  // attendi che j finisca la doorway
    while (number[j] != 0 &&
           (number[j] < number[i] ||
            (number[j] == number[i] && j < i))) ; // ordine lessicografico
  }

  /* CS */

  /* exit */
  number[i] = 0;
}
```

2.7 Calcolo del massimo: implementazione corretta

Per evitare letture inconsistenti del massimo, è fondamentale usare variabili locali temporanee e *scansione completa*:

Listing 5: Calcolo robusto di max

```
int local_max = 0;
for (int t = 1; t <= N; ++t) {
  int v = number[t];
  if (local_max < v) local_max = v;
}
number[i] = 1 + local_max;
```

2.8 Versione con numeri limitati

Le slide presentano una versione che limita i biglietti con modulo `MAXIMUM` (numeri circolari):

Listing 6: Bakery con numeri limitati (dalle slide)

```
while (true) {
  /* NCS */
  while (number[i] == 0) {  // scegli solo se non stai attendendo
    choosing[i] = true;
    number[i] = (1 + max{ number[j] | j != i }) % MAXIMUM;
    choosing[i] = false;
  }
  for (int j = 1; j <= N; ++j) if (j != i) {
    while (choosing[j]) ;
    while (number[j] != 0 &&
           (number[j] < number[i] ||
            (number[j] == number[i] && j < i))) ;
  }
  /* CS */
  number[i] = 0;
}
```

Nota: la correttezza con numeri limitati richiede attenzione alle confrontazioni circolari; qui riportiamo la forma mostrata nelle slide.

2.9 Caratteristiche evidenziate nelle slide

- I processi comunicano leggendo/scrivendo variabili condivise (come Dijkstra).
- Letture/scritture **non** sono operazioni atomiche; un lettore può leggere mentre un altro sta scrivendo.
- Nessuno riceve notifiche: il polling gestisce l'ordine.
- Ogni variabile condivisa ha un unico proprietario in scrittura; gli altri solo leggono.
- Nessun processo esegue due scritture *contemporanee*.
- I tempi di esecuzione non sono correlati.

3 Algoritmo *Bakery* in un sistema client/server (passaggio di messaggi)

Le slide presentano un'implementazione con **due thread per processo** (client/server) e canali affidabili.

3.1 Assunzioni dichiarate

- Tempo di risposta finito.
- Canali di comunicazione affidabili.

3.2 Variabili globali (per processo) e inizializzazione

Listing 7: Stato per processo (versione a messaggi)

```
/* Variabili locali globali al processo i */
int num = 0;              // "biglietto" locale
boolean choosing = false; // nella doorway?
// oltre a indirizzi IP/porte degli altri processi
```

3.3 Thread client (per il processo i)

Listing 8: Thread client: doorway + bakery via messaggi

```
while (true) {
  /* NCS */

  /* doorway */
  choosing = true;
  for (int j = 1; j <= N; ++j) if (j != i) {
    send(Pj, num);          // chiedi il numero corrente di Pj
    receive(Pj, v);
    num = max(num, v);      // accumula il max
  }
  num = num + 1;            // il mio biglietto
  choosing = false;

  /* bakery (fase di attesa ordinata) */
  for (int j = 1; j <= N; ++j) if (j != i) {
    do {                    // attesa sulla doorway altrui
      send(Pj, choosing);
      receive(Pj, v);
    } while (v == true);

    do {                    // confronto lessicografico su num
      send(Pj, v);          // richiedi numero a Pj
      receive(Pj, v);
    } while (v != 0 && (v < num || (v == num && j < i)));
  }

  /* CS */

  num = 0;                  // exit
}
```

3.4 Thread server (per il processo i)

Listing 9: Thread server: risponde a richieste

```
while (true) {
  receive(Px, message);
  if (message is a number) {
    send(Px, num);          // restituisci il tuo "biglietto"
  } else {
    send(Px, choosing);     // restituisci stato doorway
  }
}
```

4 Appendice: dettagli e cautele pratiche

- **Ordine lessicografico**: si confronta la coppia (`number[j]`, j) rispetto a (`number[i]`, i): prima il biglietto numerico, poi l'ID.
- **Doorway minimale**: il periodo tra `choosing[i]=true` e `choosing[i]=false` deve essere il più breve possibile.

- **Modello di memoria**: le slide assumono che letture/scritture semplici siano visibili secondo un modello ragionevole; su architetture reali possono servire barriere/memoria *volatile* (come indicato nelle dichiarazioni).
- **Proof sketch**: l'uso di `choosing[]` impedisce che due processi osservino simultaneamente uno stato parziale del biglietto altrui; il confronto lessicografico ordina totalmente i contendenti con biglietto diverso o uguale rompendo il pareggio con l'ID.