

Università degli Studi di Padova

Comunicazione tra processi

SCD

Anno accademico 2004/5
Corso di Sistemi Concorrenti e Distribuiti

Tullio Vardanega, tullio.vardanega@math.unipd.it

Corso di Laurea Specialistica in Informatica, Università di Padova 1/32

Università degli Studi di Padova

Comunicazione tra processi

Premesse - 1

- ❑ I processi di un sistema concorrente sono raramente indipendenti l'uno dall'altro
- ❑ La definizione delle interfacce (relazioni formali) tra le entità concorrenti di un sistema pertanto è attività fondamentale
- ❑ Occorre far riferimento ad un modello di comunicazione tra entità

Corso di Laurea Specialistica in Informatica, Università di Padova 2/32

Università degli Studi di Padova

Comunicazione tra processi

Premesse - 2

- ❑ Un modello di comunicazione può prevedere (in prima approssimazione)
 - Comunicazioni dirette tra entità attive
 - Comunicazioni indirette attraverso entità reattive di tipo passivo o protetto
- ❑ Forme "classiche" di comunicazione
 - Variabili condivise
 - Forma del tutto inadeguata se priva di agenti di controllo
 - Non garantisce l'integrità dello scambio dati se l'accesso non è indivisibile
 - Scambio messaggi

Corso di Laurea Specialistica in Informatica, Università di Padova 3/32

Università degli Studi di Padova

Comunicazione tra processi

Esempio

```
// il processo A deve accedere ad X
// Prima però deve verificarne
// lo stato di libero
if (lock == 0) {
  // X è già in uso
  // Occorre ritentare il test
}
else {
  // X è libera, allora va bloccata
  lock = 0;
  <usa (X) >;
  // e nuovamente liberata dopo l'uso
  lock = 1;
}

// il processo B deve accedere ad X
// Prima però deve verificarne
// lo stato di libero
if (lock == 0) {
  // X è già in uso
  // Occorre ritentare il test
}
else {
  // X è libera, allora va bloccata
  lock = 0;
  <usa (X) >;
  // e nuovamente liberata dopo l'uso
  lock = 1;
}
```

Questo approccio è fallimentare! Perché?

Corso di Laurea Specialistica in Informatica, Università di Padova 4/32

Università degli Studi di Padova

Comunicazione tra processi

Sincronizzazione - 1

- ❑ **Mutua esclusione**
 - Assicura che, ad ogni istante, non più di un processo abbia possesso di una risorsa (fisica o logica) condivisa
 - La sequenza di azioni che opera sulla risorsa è detta, in questo caso, sezione critica
- ❑ **Sincronizzazione condizionale**
 - Impone condizioni (logiche, di stato interno) che devono valere all'atto della sincronizzazione
 - Problema classico: il *buffer* finito condiviso nel modello produttore-consumatore

Corso di Laurea Specialistica in Informatica, Università di Padova 5/32

Università degli Studi di Padova

Comunicazione tra processi

Sincronizzazione - 2

- ❑ L'uso di sincronizzazione espone a problemi delicati
 - Non risolvibili a priori nel progetto di un linguaggio concorrente!
 - Riconoscibili e risolvibili solo nel progetto di un sistema concorrente
- ❑ Stallo (*deadlock*)
- ❑ Accodamento potenzialmente infinito (*lockout, starvation*)

Corso di Laurea Specialistica in Informatica, Università di Padova 6/32

Università degli Studi di Padova

Comunicazione tra processi

Sincronizzazione - 3

Stallo (1/2)

- **Impedisce di proseguire a tutti i processi coinvolti**
- **Richiede 4 ben riconoscibili precondizioni per aver luogo**
 - **Mutua esclusione**
(già definita)
 - **Cumulazione di risorse**
I processi possono accumulare risorse e trattenerle mentre attendono di acquisirne altre
 - **Assenza di prerilascio**
Le risorse vengono rilasciate solo volontariamente
 - **Attesa circolare**
Un processo attende una risorsa in possesso del successivo processo in catena

Corso di Laurea Specialistica in Informatica, Università di Padova 7/32

Università degli Studi di Padova

Comunicazione tra processi

Sincronizzazione - 4

Stallo (2/2)

- **4 possibili strategie per affrontare il problema**
 - **Indifferenza**
Assumere (erroneamente?) che il problema non possa verificarsi
 - **Prevenzione statica**
Richiede accertamento che progetto e realizzazione del sistema siano strutturalmente liberi da condizioni di rischio
 - **Prevenzione dinamica**
Comporta analisi dello stato di esecuzione presente e futuro del sistema per evitare esso possa entrare in una condizione di stallo (può bastare impedire strutturalmente il verificarsi di almeno una delle 4 precondizioni?)
 - **Rilevazione e trattamento**
Comporta capacità di riconoscimento del verificarsi del problema e richiede meccanismi complessi per ripristinare uno stato noto e sicuro

Corso di Laurea Specialistica in Informatica, Università di Padova 8/32

Università degli Studi di Padova

Comunicazione tra processi

Sincronizzazione - 5

Accodamento potenzialmente infinito

- **Non** si verifica in presenza di politica di accodamento FIFO
- **Si può verificare in presenza di qualunque altra politica**
 - A priorità
 - LIFO
 - Ad urgenza
- **La condizione di libertà da questo problema viene detta *fairness* (o *liveness*)**
 - Tutti i processi hanno, nel tempo, uguali opportunità di progredire
 - L'attesa attiva è incompatibile con l'accertamento di *fairness*!

Corso di Laurea Specialistica in Informatica, Università di Padova 9/32

Università degli Studi di Padova

Comunicazione tra processi

Requisiti minimi

Per l'implementazione di un linguaggio concorrente abbiamo bisogno di

- Un meccanismo pratico per evitare l'uso di attesa attiva
- **Forme di accodamento di processi (in relazione a comunicazione e sincronizzazione!) che diano garanzie di *fairness***

La correttezza funzionale del programma non deve dipendere da queste forme e meccanismi!

Corso di Laurea Specialistica in Informatica, Università di Padova 10/32

Università degli Studi di Padova

Comunicazione tra processi

Esempi classici

I filosofi a cena

- **Specifica del problema: N filosofi sono seduti ad un tavolo circolare. Ciascuno ha davanti a se cibo (spaghetti?) in un piatto, ed una posata alla propria destra. Ciascun filosofo necessita di due posate per mangiare. L'attività di ciascun filosofo alterna pasti a momenti di riflessione**

Il barbiere che dorme

- Materiale di esercitazione

Corso di Laurea Specialistica in Informatica, Università di Padova 11/32

Università degli Studi di Padova

Comunicazione tra processi

I filosofi a cena - 1

Situazione artificiosa, utile per illustrare importanti aspetti progettuali di concorrenza

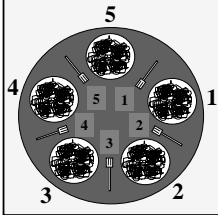
- **Condivisione risorse** → le posate
- **Mutua esclusione** → pasto condizionato al possesso di 2 posate
- **Scambio dati tra processi** → diretto tra entità attive o mediato tramite entità reattive (disponibilità delle risorse)
- **Implementazione delle risorse** → posate come risorse passive, risorse protette oppure *server*
- **Sincronizzazione condizionale** → un filosofo non può mangiare fin quando non ha 2 posate
- **Rischio di stallo** → vedi riquadro seguente

Corso di Laurea Specialistica in Informatica, Università di Padova 12/32

Università degli Studi di Padova

Comunicazione tra processi

I filosofi a cena - 2



Una non-soluzione

```
void filosofo_i(){
  while (True) {
    // medita
    prendi_forchetta(i);
    prendi_forchetta((i++)%5);
    // mangia
    posa_forchetta(i);
    posa_forchetta((i++)%5);
  }
}
```

Questo approccio incorre in tutte e 4 le precondizioni di stallo

Corso di Laurea Specialistica in Informatica, Università di Padova 13/32

Università degli Studi di Padova

Comunicazione tra processi

Possibili implementazioni - 1

Mutua esclusione con variabili condivise

```
var flag: array [0..1] of boolean;
turn: 0..1;
repeat
  flag [i] := true;
  while flag [i] do
    if turn = i then
      begin
        flag [i] := false;
        while turn /= i do no-op;
        flag [i] := true;
      end;
    end if;
  end while;
  critical section
  turn := j;
  flag [i] := false;
  remainder section
until false;
```

Algoritmo concepito da Dekker ed applicato alle sezioni critiche da Edsger Wybe Dijkstra:
flag ← 1 indica l'intenzione di entrare in sezione critica
turn consente di arbitrare l'accesso tra i processi.
 Applica a 2 processi
 i ← 0
 j ← 1

Corso di Laurea Specialistica in Informatica, Università di Padova 14/32

Università degli Studi di Padova

Comunicazione tra processi

Possibili implementazioni - 2

L'algoritmo di Dekker

- Quando P0 vuole entrare in sezione critica lo annuncia a P1 ponendo ad 1 la propria variabile di controllo (flag[0] ← 1)
- Se P1 ne è già uscito (remainder section), la sua variabile di controllo varrà 0, pertanto P0 può entrarvi
- Altrimenti occorre usare un'ulteriore variabile di controllo (turn) per arbitrare tra i 2 processi
- Se turn indica P1 (turn = 1) allora P0 rinuncia alla competizione (flag[0] ← 0) e si pone in attesa attiva che P1 abbia finito

Applicabile, ma molto pesante, a > 2 processi

Corso di Laurea Specialistica in Informatica, Università di Padova 15/32

Università degli Studi di Padova

Comunicazione tra processi

Esercizio 1

Applicare, usando un linguaggio a piacere, l'algoritmo di Dekker ad un problema concorrente concreto, verificandone il funzionamento e/o gli eventuali problemi, prima nel caso di 2 processi e, poi, con più processi

- Usare come esempio il problema dei filosofi a cena, partendo dal caso semplice con 2 soli processi "filosofo"

Corso di Laurea Specialistica in Informatica, Università di Padova 16/32

Università degli Studi di Padova

Comunicazione tra processi

I filosofi a cena - 3

Soluzione con semafori, senza stallo

```
void filosofo_i(){
  while (True) {
    // medita
    P(mutex);
    P(f[i]);
    P(f[(i+1)%5]);
    V(mutex);
    // mangia
    V(f[i]);
    V(f[(i+1)%5]);
  }
}
```

Utilizziamo un semaforo per ogni forchetta (f[1..5]) ed un semaforo per la mutua esclusione (mutex) al momento del prelievo delle forchette necessarie, così che più filosofi possano cenare simultaneamente.

Corso di Laurea Specialistica in Informatica, Università di Padova 17/32

Università degli Studi di Padova

Comunicazione tra processi

Esercizio - 2

Dato il modello di codice nel riquadro precedente, indicare

- Pregi e difetti della soluzione illustrata
- Se sia possibile una soluzione che preveda l'utilizzo del solo semaforo mutex (dunque senza un semaforo per ciascuna posata)
- Se sia possibile il caso in cui il processo i-esimo si impossessi del semaforo mutex ma trovi una delle due forchette a lui adiacenti occupate, indicando le conseguenze di tale eventualità

Corso di Laurea Specialistica in Informatica, Università di Padova 18/32

Università degli Studi di Padova

Comunicazione tra processi

Possibili implementazioni - 3

- **Mutua esclusione mediante semafori**
 - L'implementazione deve garantire che le operazioni sul semaforo (storicamente dette P e V) siano indivisibili
 - All'attesa sul semaforo deve corrispondere una forma di accodamento del processo fino alla condizione di rilascio
 - Il semaforo ha un valore di inizializzazione ed un valore massimo fissato
 - Il valore iniziale, a risorsa libera, deve consentire il primo accesso
 - Il valore massimo indica il grado di concorrenza consentito dalla risorsa (1 → mutua esclusione | > 1 → più accessi simultanei)
 - Questa tecnica dipende troppo dalla disciplina del programmatore

Corso di Laurea Specialistica in Informatica, Università di Padova 19/32

Università degli Studi di Padova

Comunicazione tra processi

Possibili implementazioni - 4

- **Mutua esclusione mediante *monitor***
 - La struttura *monitor* incapsula la risorsa condivisa e le operazioni che agiscono su di essa (dunque anche le corrispondenti sezione critiche)
 - 1974, Charles A R Hoare, "Monitors – An Operating System Structuring Concept", Communications of the ACM vol. 17, pp. 549-557 + Erratum in CACM vol. 18, p. 95.
 - La risorsa ed il suo stato sono nascosti alla vista dei processi utente
 - Il *monitor* esercita controllo sull'esecuzione delle operazioni invocate dall'esterno
 - Il compilatore (non il programmatore!) inserisce il codice necessario al controllo degli accessi

Corso di Laurea Specialistica in Informatica, Università di Padova 20/32

Università degli Studi di Padova

Comunicazione tra processi

Possibili implementazioni - 5

- **Sincronizzazione condizionale mediante *monitor***
 - Riesce a rappresentare ed a controllare condizioni logiche di accesso più sofisticate della mutua esclusione
 - Mediante variabile di condizione o segnale
 - Wait (var_condizione) → forza l'attesa del chiamante
 - Signal (var_condizione) → risveglia il processo in attesa
 - Il segnale di risveglio (Signal) non ha memoria, dunque va perso se nessuno lo ricepisce

Corso di Laurea Specialistica in Informatica, Università di Padova 21/32

Università degli Studi di Padova

Comunicazione tra processi

Il costrutto *monitor* - 1

```

monitor PC
condition non-vuoto, non-pieno;
integer contenuto;
procedure inserisci(prod : integer);
begin
  if contenuto = N then wait(non-pieno);
  <inserisci nel contenitore>;
  contenuto := contenuto + 1;
  if contenuto = 1 then signal(non-vuoto);
end;
function preleva : integer;
begin
  if contenuto = 0 then wait(non-vuoto);
  preleva := <preleva dal contenitore>;
  contenuto := contenuto - 1;
  if contenuto = N-1 then signal(non-pieno);
end;
contenuto := 0;
end monitor;
  
```

```

procedure Produttore;
begin
  while true do
  begin
    prod := produci;
    PC.inserisci(prod);
  end;
end;

procedure Consumatore;
begin
  while true do
  begin
    prod := PC.preleva;
    consuma(prod);
  end;
end;
  
```

Corso di Laurea Specialistica in Informatica, Università di Padova 22/32

Università degli Studi di Padova

Comunicazione tra processi

Il costrutto *monitor* - 2

- **Wait blocca il chiamante quando le condizioni logiche della risorsa non consentono di procedere**
 - Contenitore pieno (produttore), contenitore vuoto (consumatore)
- **Signal notifica il verificarsi della condizione attesa al primo processo bloccato, risvegliandolo**
 - Il processo risvegliato compete per l'esecuzione con il chiamante della Signal
- **Wait e Signal sono invocate in mutua esclusione**
 - Questo esclude il verificarsi di *race condition*

Corso di Laurea Specialistica in Informatica, Università di Padova 23/32

Università degli Studi di Padova

Comunicazione tra processi

Il costrutto *monitor* - 3

- **Java approssima il *monitor*, tramite classi con metodi synchronized, ma senza variabili di condizione**
 - L'attesa è sull'intero oggetto, non su una condizione
- **I metodi wait() e notify() invocati esplicitamente all'interno di metodi synchronized realizzano le condizioni evitando il verificarsi di *race condition***
 - notify() risveglia uno qualsiasi dei processi in attesa sull'oggetto
 - L'esecuzione di wait() può venire interrotta, il che va trattato esplicitamente come eccezione

Corso di Laurea Specialistica in Informatica, Università di Padova 24/32

Università degli Studi di Padova Comunicazione tra processi

Il costrutto *monitor* - 4

```

class monitor{
private int contenuto = 0;
public synchronized void inserisci(int prod){
    if (contenuto == N) blocca();
    <inserisci nel contenitore>;
    contenuto = contenuto + 1;
    if (contenuto == 1) notify();
}
public synchronized int preleva(){
    int prod;
    if (contenuto == 0) blocca();
    prod = <preleva dal contenitore>;
    contenuto = contenuto - 1;
    if (contenuto == N-1) notify();
    return prod;
}
private void blocca(){
    try{wait();}
    } catch(InterruptedException exc) {};}
}
    
```

```

static final int N = <N>;
static monitor PC = new monitor();
// ..
PC.inserisci(prod); // produttore
// ..
prod = PC.preleva(); // consumatore
    
```

Attesa e notifica sono responsabilità del programmatore

25/32

Corso di Laurea Specialistica in Informatica, Università di Padova

Università degli Studi di Padova Comunicazione tra processi

Esercizio 3

- Dato il funzionamento di `wait()` e `notify()` in Java si verifichi se**
 - Il codice mostrato nel riquadro precedente emuli correttamente il funzionamento di un *monitor* nel caso di 1 produttore ed 1 consumatore**
 - Ciò si verifichi anche nel caso di produttori e consumatori multipli**

26/32

Corso di Laurea Specialistica in Informatica, Università di Padova

Università degli Studi di Padova Comunicazione tra processi

Il costrutto *monitor* - 5

- Il *monitor* separa sincronizzazione da scambio (condivisione) dati**
- Il *monitor* non è però capace di controllare dinamicamente l'ordine degli accessi**
- Il *monitor* ha bisogno di meccanismi supplementari (`wait` & `signal`) per realizzare sincronizzazione condizionale**

27/32

Corso di Laurea Specialistica in Informatica, Università di Padova

Università degli Studi di Padova Comunicazione tra processi

Scambio messaggi - 1

- Comporta sincronizzazione (= conoscenza dello stato dell'altro) solo nel modello sincrono**
 - Il mittente ed il destinatario si attendono reciprocamente, procedendo solo a scambio avvenuto**
- Nella forma asincrona, il mittente non attende il destinatario**
 - Il destinatario non viene così a conoscere lo stato attuale del mittente**

28/32

Corso di Laurea Specialistica in Informatica, Università di Padova

Università degli Studi di Padova Comunicazione tra processi

Scambio messaggi - 2

- Mittente e destinatario devono conoscersi per indirizzarsi?**
 - Nomi unici**
 - Di processo; di casella postale; di porta; di canale
 - Tipo di messaggio (a destinazione)**
- La forma sincrona con uso di nomi unici è detta *rendezvous***
 - Processo A :: B ! Msg** **Processo B :: A ? Msg**
 - In CSP questa forma è unidirezionale**
- All'attesa di un [tipo di] messaggio può essere necessaria una *precondizione***
 - Dijkstra ha proposto che i comandi di ricezione siano selettivi, nondeterministici e dotati di "guardia"**
 - 1975, "Guarded Commands, Nondeterminacy, and Formal Derivation of Programs", CACM, vol. 18(8), pp. 453-457**

29/32

Corso di Laurea Specialistica in Informatica, Università di Padova

Università degli Studi di Padova Comunicazione tra processi

Scambio messaggi - 3

- La "guardia" è una espressione Booleana la cui verità abilita l'esecuzione del comando ad essa associato**

```

select
    Guard_1 => Statement_1;
or
    Guard_2 => Statement_2;
or
    ..
or
    Guard_N => Statement_N;
end select;
    
```

Se più di una guardia è vera simultaneamente, ne viene scelta una **nondeterministicamente**

30/32

Corso di Laurea Specialistica in Informatica, Università di Padova

Università degli Studi di Padova

Comunicazione tra processi

Scambio messaggi - 4

- Lo scambio dati può utilmente essere bidirezionale senza richiedere 2 messaggi
- Il primo destinatario allora offre un "luogo di entrata" (entry) presso il quale ricevere parametri *in*, elaborarli e restituire valori su parametri *out*

```
Guard =>  
  accept Service ( in ... out ...) do  
    ..  
  end;
```

- Con questa forma l'invocante nomina il destinatario ed il servizio (= canale) ma non viceversa
 - Asimmetria di denominazione

Corso di Laurea Specialistica in Informatica, Università di Padova 31/32

Università degli Studi di Padova

Comunicazione tra processi

Scambio messaggi - 5

- **Forma sincrona e forma asincrona sono duali: l'una può essere utilizzata per ottenere l'altra**
 - **Sincrona → Asincrona**
Introdurre un'entità intermedia tra mittente e destinatario rende la loro comunicazione asincrona
 - Al costo aggiuntivo dell'entità intermedia (minor costo se non attiva!)
 - **Asincrona → Sincrona**
Accoppiare Send (conferma) a Receive dal lato destinatario e Receive a Send dal lato mittente forza sincronizzazione

Corso di Laurea Specialistica in Informatica, Università di Padova 32/32