

Comunicazione e sincronizzazione tra processi Indice

- 1 Meccanismi
 - 1.1 *Monitor*
 - 1.2 Scambio di messaggi
 - 1.3 Barriere
- 2 Problemi classici
- 3 Stallo

Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 1

Comunicazione e sincronizzazione *Monitor - 1*

- La soluzione mediante semafori è di uso difficile e rischioso
 - Un posizionamento incauto delle P può causare situazioni di blocco infinito (*deadlock*) ed esecuzioni erronee di difficile verifica (*race condition*)
 - Non è desiderabile lasciare all'utente il controllo di strutture così delicate

Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 2

Esempio 1

```
#define N ... /* posizioni del contenitore */
typedef int semaforo;
semaforo mutex = 1; /* P decrementa, V incrementa, 0 blocca */
semaforo non-pieno = N;
semaforo non-vuoto = 0;

void produttore(){
    int prod;
    while(1){
        prod = produci();
        P(&non-pieno);
        P(&mutex);
        inserisci(prod);
        V(&mutex);
        V(&non-vuoto);
    }
}

void consumatore(){
    int prod;
    while(1){
        P(&non-vuoto);
        P(&mutex);
        prod = preleva();
        V(&mutex);
        V(&non-pieno);
        consuma(prod);
    }
}
```

Il corretto ordinamento di P e V è critico!

Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 3

Comunicazione e sincronizzazione *Monitor - 2*

- Dall'esempio 1 si nota come un diverso ordinamento delle P nel codice utente potrebbe causare situazioni di blocco infinito (*deadlock*)


```
P(&mutex); /* accesso esclusivo al contenitore */
P(&non-pieno); /* attesa spazi nel contenitore */
```
- Il consumatore non può accedere al contenitore per prelevare prodotti, facendo spazio per altri ...

Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 4

Comunicazione e sincronizzazione *Monitor - 3*

- Linguaggi evoluti di alto livello (e.g.: Concurrent Pascal, Ada, Java) offrono una struttura esplicita di controllo delle regioni critiche, originariamente detta *monitor* (Hoare, '74; Brinch-Hansen, '75)
- Il *monitor* definisce la regione critica
- Il compilatore (non il programmatore!) inserisce il codice necessario al controllo degli accessi

Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 5

Comunicazione e sincronizzazione *Monitor - 4*

- Un *monitor* è un aggregato di sottoprogrammi, variabili e strutture dati
- Solo i sottoprogrammi del *monitor* possono accedervi le variabili interne
- Solo un processo alla volta può essere attivo entro il *monitor*
 - Proprietà garantita dai meccanismi del supporto a tempo di esecuzione del linguaggio di programmazione concorrente, il cui codice è inserito dal compilatore nel programma eseguibile

Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 6

Comunicazione e sincronizzazione

Monitor - 5

- La sola garanzia di mutua esclusione può non bastare ad affrontare il problema
- Due procedure operanti su variabili speciali (non contatori!) dette *condition variables*, consentono di modellare condizioni logiche specifiche del problema
 - Wait**(<condizione>) /* forza l'attesa del chiamante */
 - Signal**(<condizione>) /* risveglia il processo in attesa */
- Il segnale di risveglio (*signal*) non ha memoria
 - Va perso se nessuno lo attende

Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 7

Esempio 2

```

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;

```

Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 8

Comunicazione e sincronizzazione

Monitor - 6

- Il meccanismo di *wait* permette di bloccare il processo chiamante quando le condizioni logiche della risorsa non gli consentono di procedere
 - E.g.: Contenitore pieno per il produttore; Contenitore vuoto per il consumatore
- Il meccanismo di *signal* notifica il verificarsi della condizione attesa al (primo) processo bloccato, risvegliandolo
 - Il processo risvegliato compete con il chiamante della *signal* per il possesso della CPU
- Wait* e *signal* sono invocate in mutua esclusione
 - Non si può verificare *race condition*

Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 9

Comunicazione e sincronizzazione

Monitor - 7

- Java supporta monitor, mediante classi con metodi **synchronized**, ma senza *condition variables*
- Le primitive **wait()** e **notify()** invocate all'interno di metodi **synchronized** evitano il verificarsi di *race condition*
 - In realtà, il metodo **wait()** può venire interrotto, e l'interruzione va trattata come eccezione!

Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 10

Esempio 3

```

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

```

Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 11

Comunicazione e sincronizzazione

Monitor - 8

- In ambiente locale si hanno 3 possibilità
 - Linguaggi concorrenti con supporto esplicito per strutture monitor (alto livello)
 - Linguaggi sequenziali senza alcun supporto per monitor o semafori
 - Uso di semafori tramite strutture primitive del sistema operativo e chiamate di sistema (basso livello)
 - Implementazione di semafori, in linguaggio assembler, senza alcun supporto dal sistema operativo (bassissimo livello)
- Monitor e semafori non sono utilizzabili per realizzare scambio di informazione tra elaboratori

Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 12

Comunicazione e sincronizzazione Scambio messaggi - 1

- Utilizzabile allo stesso modo sia in locale che in remoto
 - Caratteristica assai desiderabile
- Due dimensioni di scelta di modello
 - Sincrono / asincrono
 - Mediante chiamate di sistema / costrutti di linguaggio

Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 13

Comunicazione e sincronizzazione Scambio messaggi - 2

- Primitive di scambio messaggi
 - **Send**(destinatario, messaggio) // per copia o per riferimento
 - **Receive**(mittente, messaggio) // per copia o per riferimento
- Struttura di comunicazione
 - Canale simmetrico
 - 1 mittente : 1 destinatario
 - Canale asimmetrico
 - 1 mittente : N destinatari ; N mittenti : 1 destinatario
 - Casella postale (*mailbox*)
 - N mittenti : M destinatari

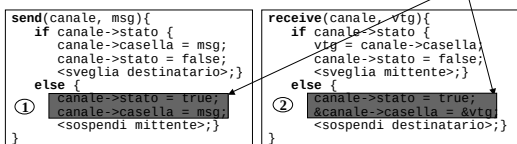
Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 14

Comunicazione e sincronizzazione Scambio messaggi - 3

- Comunicazione sincrona (*rendezvous*)
 - Mittente e destinatario sono entrambi titolari di uno stesso canale di comunicazione
 - Canale tipicamente simmetrico



Comunicazione e sincronizzazione

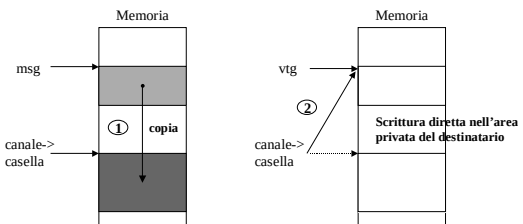
Architettura degli elaboratori 2 - T. Vardanega

Pagina 15

Comunicazione e sincronizzazione Scambio messaggi - 4

Visione del mittente

Visione del destinatario



Casella (*mailbox*) ad 1 posizione

Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 16

Comunicazione e sincronizzazione Scambio messaggi - 5

- Comunicazione asincrona
 - Il mittente deposita il messaggio e procede
 - Il destinatario attende la consegna per procedere
 - Richiede una casella capace di N posizioni, per N messaggi che il mittente può depositare prima di un prelievo
 - N è il grado di asincronia
 - Il mittente scrive direttamente in vtg se il destinatario è in attesa, oppure nella prima posizione libera, altrimenti si sospende
 - Rischio di attesa infinita (meglio sovrascrittura su casella strutturata ad anello)

Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 17

Comunicazione e sincronizzazione Scambio messaggi - 6

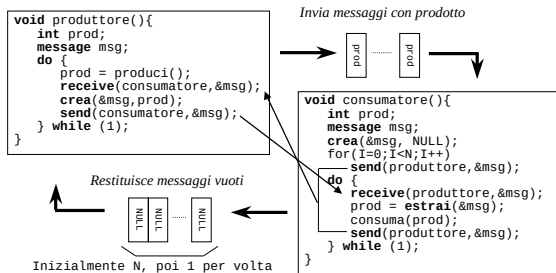
- In presenza di linee di comunicazione inaffidabili
 - Conferma di ricezione (*acknowledgment*)
 - Per prevenire la perdita del messaggio inviato
 - Controllo della duplicazione
 - Per prevenire la perdita della conferma di ricezione
- Identificazione non ambigua di mittente e destinatario
 - Denominazione (*naming*)
 - Autenticazione (*authentication*)

Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 18

Esempio 4



- Modalità di comunicazione asincrona, per riferimento
- I messaggi non ricevuti sono accodati nel canale di comunicazione

Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 19

Comunicazione e sincronizzazione
Barriera

- Consente di sincronizzare gruppi di processi
 - Attività cooperative suddivise in fasi ordinate
- La barriera blocca tutti i processi che la raggiungono fino all'arrivo dell'ultimo
 - Si applica indistintamente ad ambiente locale e distribuito
- Non comporta scambio di messaggi esplicito

Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 20

Comunicazione e sincronizzazione
Problemi classici

- Metodo per valutare l'efficacia e l'eleganza di modelli e meccanismi per la comunicazione e la sincronizzazione tra processi
 - **Filosofi a cena** : accesso esclusivo a risorse limitate
 - **Lettori e scrittori** : accessi concorrenti a basi di dati
 - **Barbieri che dorme** : prevenzione di *race condition*
- Problemi pensati per rappresentare tipiche situazioni di rischio
 - Stallo con blocco (*deadlock*), stallo senza blocco (*starvation*)
 - Esecuzioni non predicibili (*race condition*)

Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 21

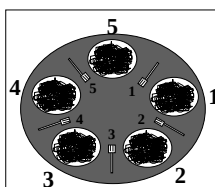
Comunicazione e sincronizzazione
Filosofi a cena - 1

- N filosofi sono seduti ad un tavolo circolare
- Ciascuno ha davanti a se 1 piatto ed 1 posata alla propria destra
- Ciascun filosofo necessita di 2 posate per mangiare
- L'attività di ciascun filosofo alterna pasti a momenti di riflessione

Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 22

Comunicazione e sincronizzazione
Filosofi a cena - 2Soluzione A con stallo (*deadlock*)

```
void filosofo_i_esimo(){
do {
medita();
P(f[i]);
P(f[(i+1)%N]);
mangia();
V(f[(i+1)%N]);
V(f[i]);
} while (1);
}
```

L'accesso alla prima
forchetta non garantisce
l'accesso alla seconda!

Ogni forchetta modellata come un semaforo binario

Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 23

Comunicazione e sincronizzazione
Filosofi a cena - 3Soluzione B con stallo (*starvation*)

```
void filosofo_i_esimo(){
OK = 0;
do {
medita();
do {
P(f[i]);
if (f[(i+1)%N] == 0) {
V(f[i]);
sleep(T);
}
else {
P(f[(i+1)%N]);
OK = 1;
}
while (!OK);
mangia();
V(f[(i+1)%N]);
V(f[i]);
} while (1);
}
```

Un'attesa a durata fissa
difficilmente genera una
situazione differente!

Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 24

Comunicazione e sincronizzazione *Filosofi a cena - 4*

- Il problema ammette varie soluzioni
 - Per esempio, utilizzare in soluzione A un semaforo a mutua esclusione per incapsulare gli accessi ad *entrambe* le forchette
 - Funzionamento garantito
 - Alternativamente, in soluzione B, ciascun processo potrebbe attendere un tempo casuale invece che fisso
 - Funzionamento non garantito
 - Algoritmi sofisticati, con maggiore informazione sullo stato di progresso del vicino e maggior coordinamento delle attività
 - Funzionamento garantito

Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 25

Comunicazione e sincronizzazione *Stallo: condizioni necessarie e sufficienti*

- Accesso esclusivo a risorsa condivisa
- Accumulo di risorse
 - I processi possono accumulare nuove risorse senza doverne rilasciare altre
- Inibizione di prerilascio
 - Il possesso di una risorsa deve essere rilasciato volontariamente
- Condizione di attesa circolare
 - Un processo attende una risorsa in possesso del successivo processo in catena

Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 26

Comunicazione e sincronizzazione *Stallo: prevenzione - 1*

- Almeno tre strategie per affrontare lo stallo
 - **Prevenzione**
 - Impedire almeno una delle condizioni precedenti
 - **Riconoscimento e recupero**
 - Ammettere che lo stallo si possa verificare, ma essere in grado di riconoscerlo e possedere una procedura di recupero (sblocco)
 - **Indifferenza**
 - Considerare molto bassa la probabilità di stallo e non prendere alcuna precauzione contro di esso
 - Che succede se esso si verifica?

Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 27

Comunicazione e sincronizzazione *Stallo: prevenzione - 2*

- Prevenzione sulle condizioni necessarie e sufficienti
 - Accesso esclusivo alla risorsa
 - Alcune risorse non consentono alternative
 - Accumulo di risorse
 - Assai ardua da eliminare
 - Inibizione del prerilascio
 - Alcune risorse non consentono alternative
 - Attesa circolare
 - Di riconoscimento difficile ed oneroso

Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 28

Comunicazione e sincronizzazione *Stallo: prevenzione - 3*

- Prevenzione sulle richieste di accesso
 - Ad ogni richiesta di accesso, verificare se questa può portare allo stallo
 - In caso affermativo non è però chiaro cosa convenga fare
 - La verifica è un onere pesante ad ogni richiesta
 - Una alternativa è richiedere preventivamente a tutti i processi quali risorse essi richiederanno così da ordinarne l'attività in maniera conveniente

Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 29

Comunicazione e sincronizzazione *Stallo: prevenzione - 4*

- Riconoscimento
 - Assai oneroso
 - Occorre bloccare periodicamente l'avanzamento del sistema per analizzare lo stato di tutti i processi e verificare se quelli in attesa costituiscono una lista circolare chiusa
 - Lo sblocco di uno stallo comporta la terminazione forzata di uno dei processi in attesa
 - Il rilascio delle risorse liberate sblocca la catena di dipendenza circolare

Comunicazione e sincronizzazione

Architettura degli elaboratori 2 - T. Vardanega

Pagina 30