SPIN: Comunicazione tra N processi attraverso un unico canale condiviso

Sapienza Università di Roma
Laurea specialistica in Ingegneria Informatica
Tesina del corso di
Metodi formali nell'ingegneria del software
AA 2006-2007

Prof. Toni Mancini

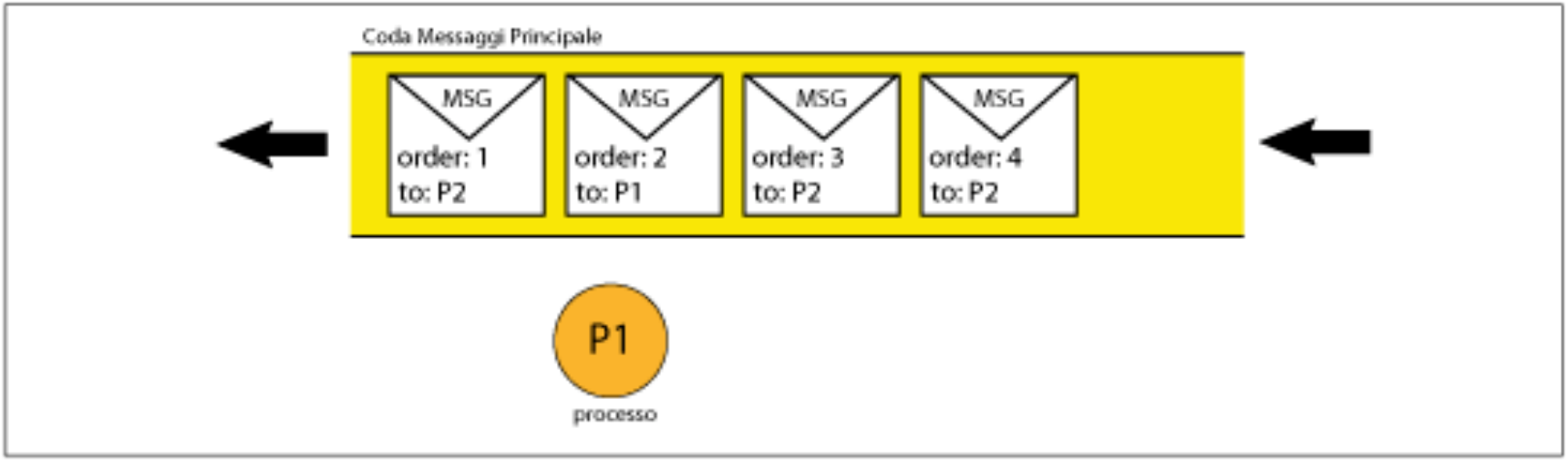
A cura di
Francesco Cusmai
Marta Monteleone
Silvia Orsi

INTRODUZIONE

- Composizione del sistema
  - Più processi eseguiti parallelamente
  - Comunicazione tramite una coda di messaggi condivisa
- Problema
  - Protocollo di comunicazione tra processi
  - Concorrenza di accesso sulla coda
  - Mantenere l'ordine di esecuzione delle azioni

PROBLEMA

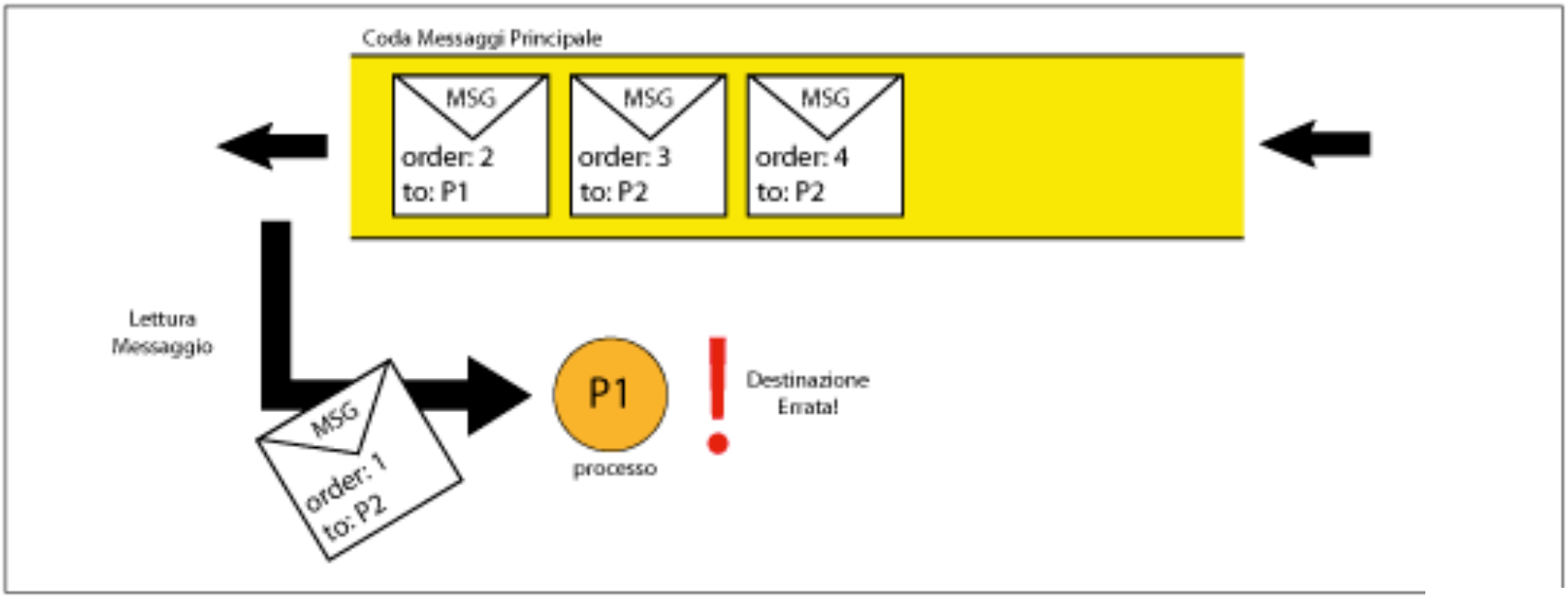
- Problema: concorrenza su una coda condivisa



  - Coda di messaggi condivisa
  - Più processi paralleli

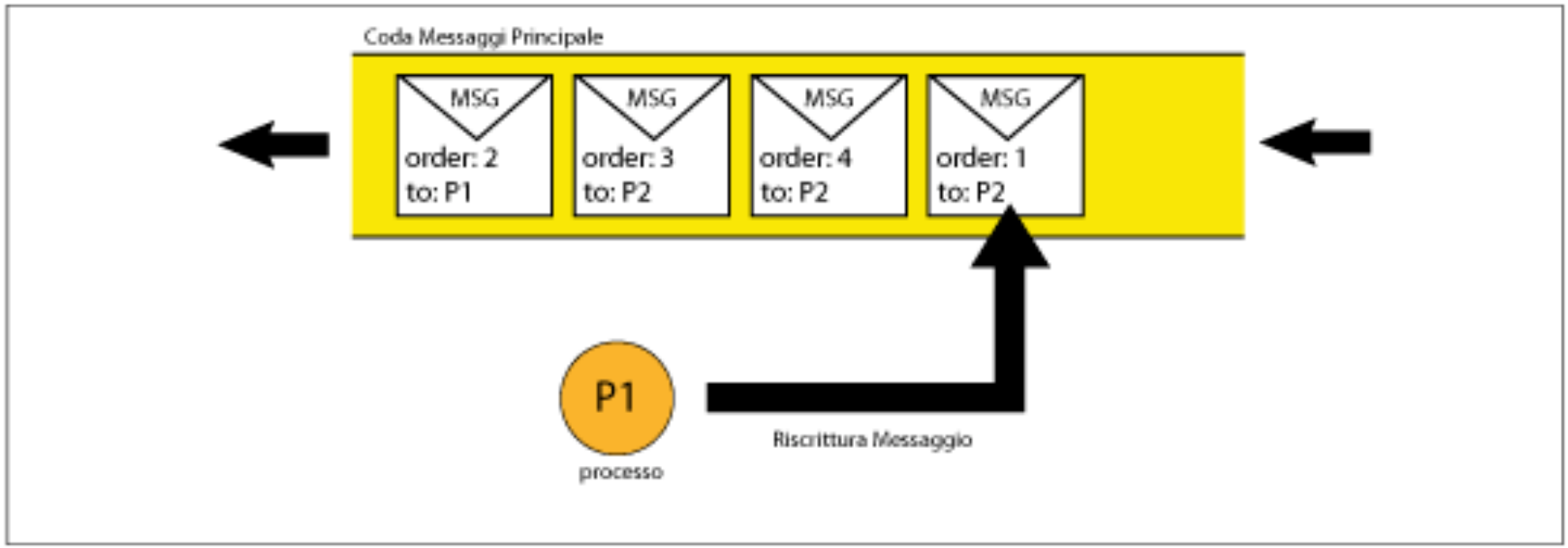
PROBLEMA

- Un processo legge un messaggio che non è indirizzato a lui…



PROBLEMA

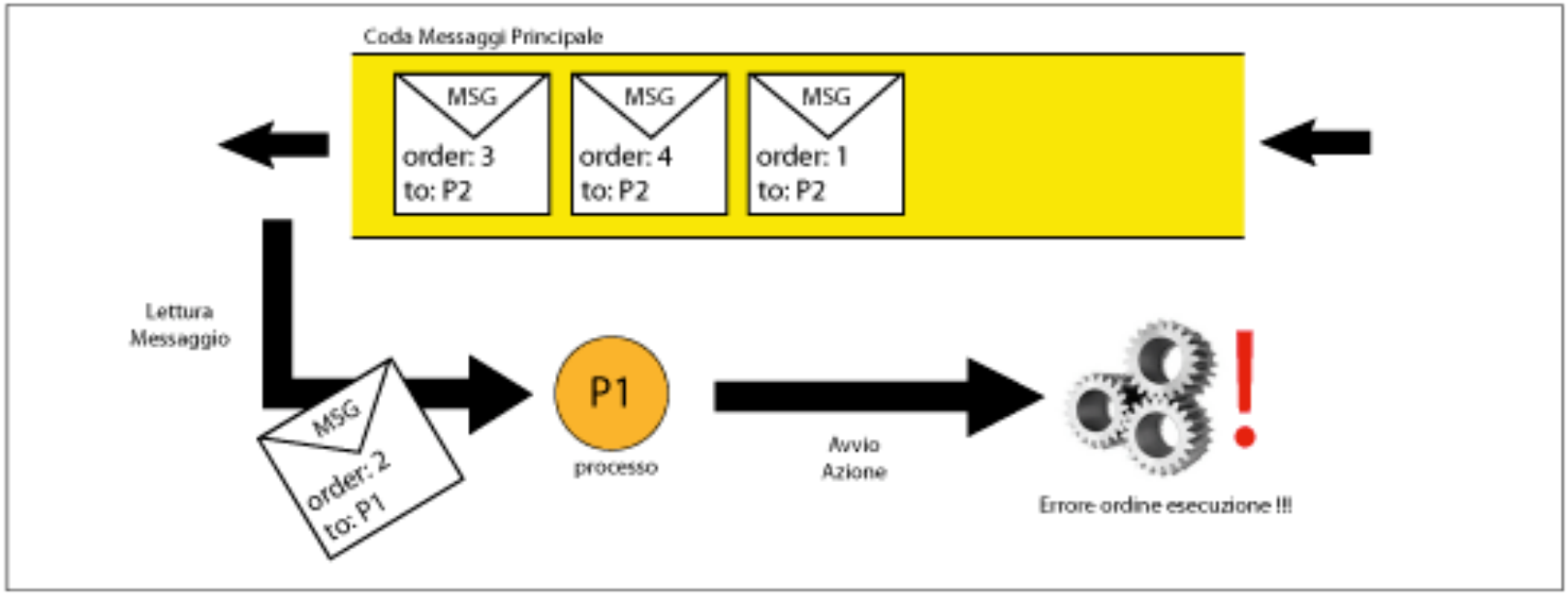
- …il messaggio viene rimesso in coda mantenendo il suo valore d'ordine di esecuzione.



La coda ora ha gli elementi che non sono più in ordine!

PROBLEMA

- Il processo legge il successivo messaggio, vede che è destinato a lui ed esegue l'azione!



Se è verificato un **errore**!!!

Non si è rispettato l'ordine di esecuzione!

LA GESTIONE DELLA CODA

- *nomecanale?var1,var2,…,varn,TIPOMESSAGGIO*
- *nomecanale?eval(var1),eval(var2),..,eval(varn), TIPOMESSAGGIO*
- *nomecanale??MSG*
- *nomecanale?[MSG]*
- *nomecanale?<MSG>*
- *nomeCanale!!var1,var2,…varn*

LA GESTIONE DELLA CODA

- *empty(nomecanale) : true se nomecanale è vuoto, false altrimenti.*
- *nempty(nomecanale) : false se nomecanale è vuoto, true altrimenti*

ATOMICITA' E TIMEOUT

- *atomic{sequenza istruzioni}*
- *d_step{sequenza di istruzioni}*
- *timeout*

DIAGRAMMA UML DEL SISTEMA PONTE

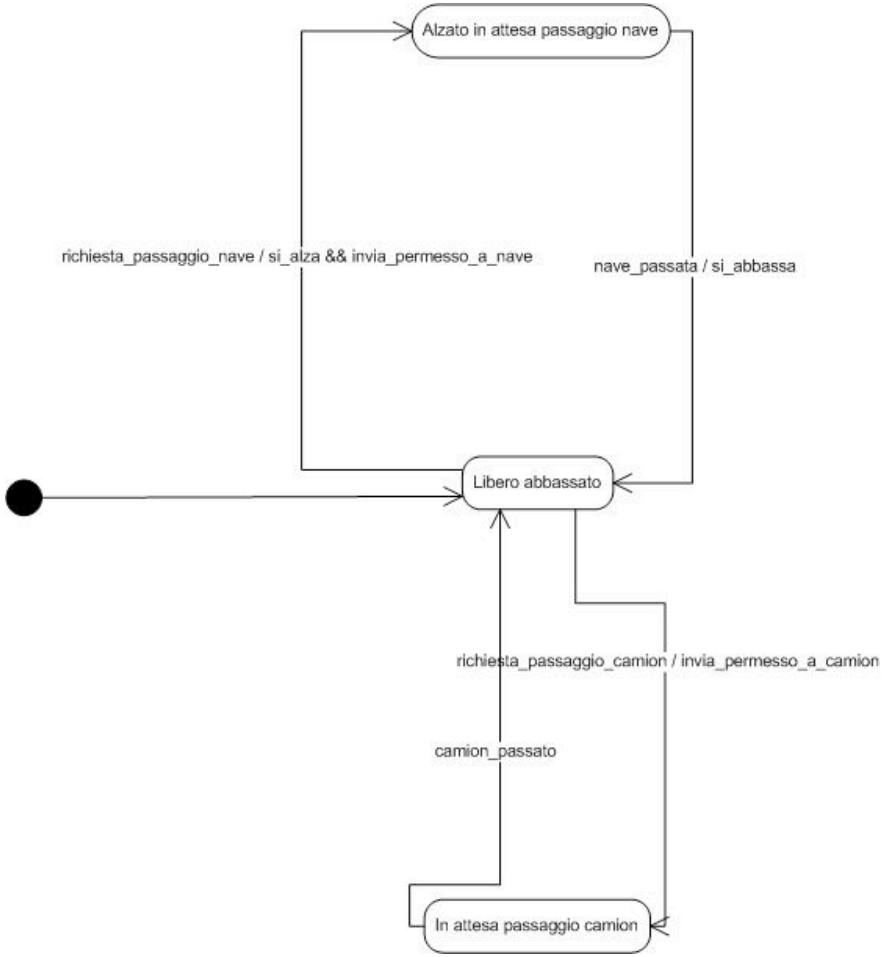


DIAGRAMMA UML DEL SISTEMA CAMION

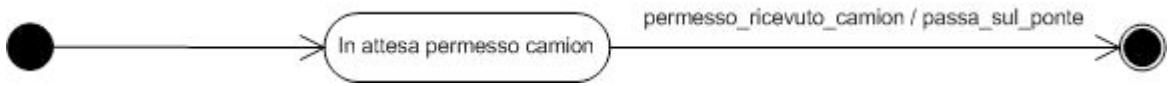
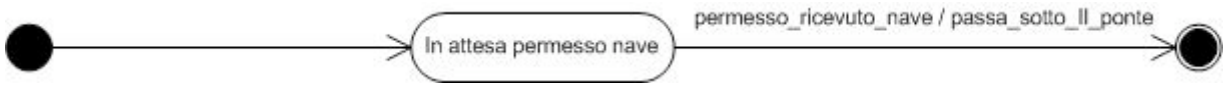


DIAGRAMMA UML DEL SISTEMA NAVE



ALGORITMO 1 : caratteristiche

- Risoluzione del problema:
  - Suddivisione del sistema in processi
  - Passaggio dal diagramma a stato a codice
  - Interazione tra i processi
  - Esecuzione delle azioni in ordine
- Passi fondamentali dell'algoritmo:
  - Definizione Variabili e Costanti
  - Definizione dei Messaggi
  - Definizione dei Processi

ALGORITMO 1 : variabili e costanti

- Variabili
  - Per identificare lo stato degli oggetti del sistema
    - posizione_ponte
  - Variabili utili per i vincoli e per le condizioni (se necessario)
    - discarica_capienza o discarica_occupata
- Definizioni delle Costanti
  - Una per ogni stato dei processi
    - PONTE_LIBERO_ABBASSATO
  - Una per ogni evento del processi
    - PONTE_RICHIESTA_CAMION
  - Una per identificare ogni processo
    - ID_PONTE

ALGORITMO 1 : messaggi

- Definizione dei Messaggi

  Ogni MESSAGGIO è definito da 4 interi:

  – int dest; …… Destinazione
  – int sorg; …… Sorgente
  – int msg_order; …… Ordine esecuzione
  – int msg; ……Tipo evento

ALGORITMO 1 : ordine esecuzione

- Per essere processato il messaggio deve possedere le seguenti caratteristiche:

  1. deve essere il messaggio più vecchio tra tutti quelli presenti nella coda;
  2. devono essere contemporaneamente soddisfatte le condizioni.

- Il messaggio più vecchio può essere ricavato tramite l'intero "**msg_order**" che è univoco per ogni messaggio. Ogni nuovo messaggio inserito in coda ha un valore del "msg_order" più grande dell'ordine massimo esistente fino a quel momento. Se viene riscritto un messaggio nella coda questo mantiene lo stesso order (tranne nel caso di riscrittura di un messaggio non atteso in un certo stato)!

- Si utilizzano delle variabili globali "**current_order**" e "**max_order**" per identificare rispettivamente l'ordine del messaggio che si deve processare in questo momento (si presuppone che sia il più piccolo) ed il numero massimo dell'ordine finora assegnato (cosi da sapere sempre quale è il prossimo order da assegnare).

ALGORITMO 1 : esempio

- Definizione processo (1)

```
proctype Processo_NOME_DIAGRAMMA_STATI ( int Process_Id )
{
  inizializzazione delle variabili locali
  creazione variabile di stato interno del processo
  LOOP_LETTURA  {
     lettura del messaggio
     if ( verifica di appartenenza del messaggio )
     { riscrittura del messaggio nella coda di sistema }
     else{
          if ( messaggio.ordine > sistema.ordine_corrente )
          { riscrittura del messaggio nella coda di sistema }
          else {
               if ( gestito( messaggio.evento ) )
               { salto alla transazione di stato associata (codice sottostante)  }
               else
               {  riscrittura del messaggio nella coda incrementando l'ordine
messaggio e quello corrente) } } } }
…
```

ALGORITMO 1 : esempio

- Definizione processo (2)

/* eseguo l'azione, la transazione, modifico lo stato del sistema e torno nel loop di lettura */
Label di gestione delle transazioni di stato
{ if (**not condizioni transazione soddisfatte**)
{ gestisco l' errore }
else {
eseguo **azione**
cambio le variabili esterne
invio i messaggi agli altri processi per **l'interazione**
cambiamento di stato
indico di passare al prossimo messaggio da gestire (incrementando l'ordine di esecuzione : current_order++)}
salto al LOOP_LETTURA o salto allo stato di terminazione del processo (salta in STATO_FINE) }
STATO_FINE: {
gestisce la terminazione del processo }
} /* **Fine definizione processo** */

ALGORITMO 2 : caratteristiche

- Scrittura in coda in modo ordinato

  Out!!0,0,pd,2,MSG

- Definizione dei messaggi

  { int , int , int ,int, mtype }
  */*priorità,mittente,destinatario,tipo:nave o camion?*/*

- Priorità dei messaggi

  0 indica priorità privilegiata(processo che si sta servendo)

ALGORITMO2: esempio servente

ciclo attesaRichieste :

::se la coda non è vuota esce

::se non vi sono più i processi da gestire termina

if il processo da servire è un camion salta a **passaggioCamion**
else contrario a **passaggioNave**

passaggioCamion: send ordinata del messaggio “permesso transito” per il camion

passaggioNave: send ordinata del messaggio “permesso transito” per la nave

ciclo attendiPassaggio: /*aspetta che il servizio venga erogato*/

if la coda non è vuota si legge e si esce

if **::**la priorità del messaggio letto non è 0 lo si rimette in coda in modo ordinato

::la destinazione non è corretta come sopra

else si esce e si torna ad aspettare nuove richieste

ALGORITMO2: esempio cliente

invioRichiesta: invia messaggio richiesta servizio (scrittura ordinata) e salta a **attendiPermesso**

ciclo attendiPermesso:

se la coda non è vuota leggo altrimenti continuo ad aspettare

if ::la priorità non è 0 rimetto in coda il messaggio in modo ordinato e torno al ciclo **attendiPermesso**

::il messaggio non è per me rimetto in coda il messaggio in modo ordinato e torno al ciclo **attendiPermesso**

::il mittente non è corretto rimetto in coda il messaggio in modo ordinato e torno al ciclo **attendiPermesso**

else salta a **passa**

passa: scrivo in coda che sono stato servito (scrittura ordinata) e termino

ALGORITMO 3 : caratteristiche

- Verifica presenza del messaggio nella coda

  Out?[0,0,eval(me),2,MSG]

- Scrittura in coda in modo ordinato

  Out!!0,0,pd,2,MSG

- Definizione dei messaggi

  { int , int , int ,int, mtype }

  */*priorità,mittente,destinatario,tipo:nave o camion?*/*

- Priorità dei messaggi

  0 indica priorità privilegiata(processo che si sta servendo)

ALGORITMO3 : esempio servente

ciclo attendiRichieste: ::se la coda non è vuota salta ad **attendiMex**,
::se non visono più processi da servire termina

ciclo attendiMex: ::se nella coda c'è il messaggio atteso allora si va a **cicloLettura**

cicloLettura: si legge dalla coda in modo atomico :
if tipo è camion **then** salta a **passaCamion**
else salta a **passaNave**

passaCamion: send ordinata del messaggio "permesso transito" per il camion
passaNave: send ordinata del messaggio "permesso transito" per la nave

attendiPassaggio: si aspetta che il servizio sia erogato e se la coda non è vuota si salta ad **attendiMex2**
attendiMex2: si controlla se il messaggio è presente in coda,in caso affermativo si salta a **cicloLettura2**

cicloLettura2:
se nella coda è presente il messaggio che segnala che il servizio è stato erogato si torna a **attendiRichieste**

ALGORITMO3: esempio cliente

invioRichiesta: scrive nella coda la richiesta di servizio (in modo ordinato) e salta ad **attendiPermesso**

attendiPermesso: si aspetta finchè un messaggio non è scritto in coda, a quel punto si salta ad **attendiMex**

attendiMex:ciclo per controllare se il messaggio desiderato è in coda, se si si salta a **cicloLettura**

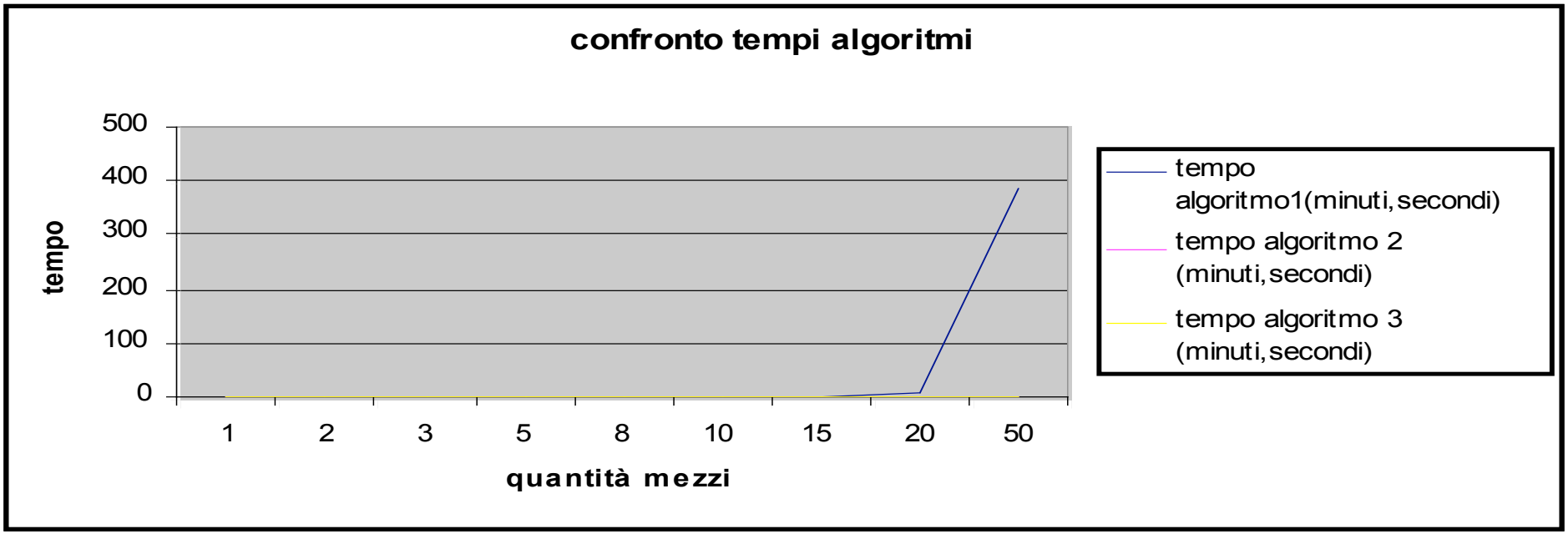
cicloLettura: quando si legge il messaggio che indica che è il proprio turno si salta a **passaSottoPonte**

passaSottoPonte:si scrive in coda in modo ordinato che si è stati serviti e si termina

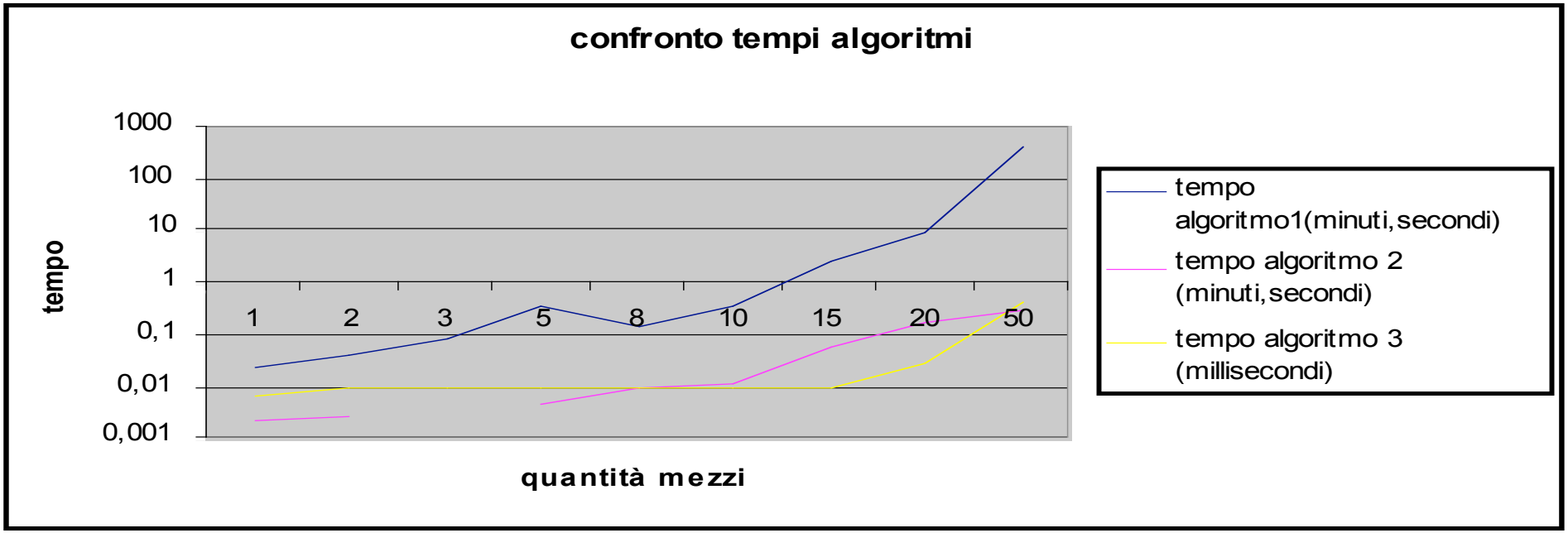
TEMPI ESECUZIONE ALGORITMI

N° processi totali	Tempo es. 1 (minuti,secondi,millisecondi)	Tempo es. 2 (minuti,secondi,millisecondi)	Tempo es. 3 (minuti,secondi,millisecondi)
1	*0.0.244*	*0.0.23*	*0.0.67*
2	*0.0.367*	*0.0.27*	*0.0.100*
3	*0.0.741*	*0.0.32*	*0.0.100*
5	*0.3.147*	*0.0.43*	*0.0.100*
8	*0.13.968*	*0.0.86*	*0.0.100*
10	*0.33.077*	*0.0.113*	*0.0.100*
15	*2.4.276*	*0.0.551*	*0.0.100*
20	*8.8.903*	*0.1.645*	*0.0.283*
50	*386.0.624*	*0.29.356*	*0.0.600*
100	-	*6.27.58*	*0.1.450*
200	-	*87.23.063*	*0.4.400*

TEMPI ESECUZIONE ALGORITMI



TEMPI ESECUZIONE ALGORITMI (scala logaritmica)



PROPRIETA' DEGLI ALGORITMI

PROPRIETA'	**ALGORITMO1**	**ALGORITMO2**	**ALGORITMO3**
Linee di codice	***S*** *:In termini di linee di codice è il più complesso, necessita di molte variabili e la ricezione dei messaggi è molto complessa (bisogna estrarre il messaggio affiorante,controllare tipo e destinatario ed eventualmente reinserirlo).*	***V*** *:Meno linee di codice rispetto al precedente in quanto sono richieste poche variabili e la riscrittura dei messaggi in coda avviene in modo più semplice perchè la coda è ordinata(tuttavia si estrae ancora il messaggio affiorante e lo si controlla).*	***V*** *:Come il precedente ma le linee di codice sono ancora meno. Il messaggio affiorante dalla coda è estratto da un particolare processo sse il processo in questione è in attesa esattamente di quel messaggio.*

PROPRIETA' DEGLI ALGORITMI

Ordine dei messaggi	***V*** *:L'ordinamento dei messaggi viene mantenuto gestendo un campo order del messaggio stesso con un algoritmo esterno.*	***V****:La coda è ordinata perché i messaggi vengono inseriti in modo ordinato sulla base dei campi del messaggio stesso.*	***V****:La coda è ordinata perché i messaggi vengono inseriti in modo ordinato sulla base dei campi del messaggio stesso.*
Carico	***S****:Il carico maggiore (cioè la ricezione) è sul ricevente.*	***S****:Il carico maggiore (cioè la ricezione) è sul ricevente.*	***S****:Il carico maggiore (cioè la ricezione) è sul ricevente.*
Starvation	***V****:Tutti i processi prima o poi vengono serviti.*	***V****:Tutti i processi prima o poi vengono serviti.*	***V****:Tutti i processi prima o poi vengono serviti.*
Numero di processi serviti	***S****:Con un solo canale fino a 202 circa.*	***S****:Con un solo canale fino a 202 circa.*	***S****:Con un solo canale fino a 202 circa.*

PROPRIETA' DEGLI ALGORITMI

Inserimenti e reinserimenti	***S**:Molte estrazioni e reinserimenti in coda.*	***S**:Molte estrazioni e reinserimenti in coda.*	***V**:Il numero di estrazioni e reinserimenti è ridottissimo rispetto ai precedenti (un messaggio viene estratto solo se è del tipo desiderato).*
Ordine di servizio	***V**:I processi vengono serviti esattamente nell'ordine in cui hanno fatto la richiesta.*	***S**:Non c'è garanzia che due processi dello stesso tipo vengano serviti nell'ordine in cui hanno fatto la richiesta anche se prima o poi verranno serviti.*	***S**:Non c'è garanzia che due processi dello stesso tipo vengano serviti nell'ordine in cui hanno fatto la richiesta anche se prima o poi verranno serviti.*

PROPRIETA' DEGLI ALGORITMI

Tempi di esecuzione per il numero massimo di processi serviti	***S**:Circa 386m,624msec(il numero è solo di 50 a causa dell'eccessivo tempo richiesto)*	***S**:Circa 87m,23s,063msec.*	***V**:Circa 4s,400msec*
Proprietà LTL verificate	***V**:Se ci sono due processi diversi vengono serviti in ordine. Se il ponte è alzato un camion non passa. Se il ponte è abbassato una nave non passa.*	***S**:Se ci sono quattro processi diversi vengono serviti in ordine(con più processi questa proprietà non si è certi che venga garantita). Se il ponte è alzato un camion non passa. Se il ponte è abbassato una nave non passa.*	***S**:Formalmente nessuna.*

Sviluppi futuri

- Politica con master in scrittura e lettura
- Rilevamento perdita di messaggi
- Gestione interattiva dell'arrivo dei mezzi
- Gestione evoluta delle condizioni
- Verifica di eventuali vincoli
- Gestione di una lista di attesa
- Ottimizzazione degli algoritmi per migliorare i tempi di computazione e le prestazioni generali del sistema

RIFERIMENTI

- Sito di spin : http://spinroot.com/spin/whatispin.html
- SPIN Beginners' Tutorial ,Theo Ruys
- Advanced SPIN Tutorial ,Theo Ruys & Gerard Holzmann
- Tesina del corso di MFIS "Modellazione e verifica formale di algoritmi per la mutua esclusione in ambiente distribuito", F. Delle Fave & V.Gheri
- Tesina del corso di MFIS "Verifica formale del software:SPIN", D.Nifosi