



## Correttezza temporale

# SCD

Anno accademico 2010/11  
Corso di Sistemi Concorrenti e Distribuiti

Tullio Vardanega, [tullio.vardanega@math.unipd.it](mailto:tullio.vardanega@math.unipd.it)

Corso di Laurea Magistrale in Informatica, Università di Padova 1/17



Correttezza temporale

## Premesse – 1

- ❑ La correttezza funzionale di un programma concorrente non dovrebbe dipendere dall'ordine d'esecuzione dei processi
- ❑ Occorre accertare che il non-determinismo dell'esecuzione non causi *deadlock* o *livelock*
- ❑ Il modello di concorrenza visto fin qui è non-deterministico rispetto a
  - Ordinamento d'esecuzione dei processi (*dispatching*)
  - Scelta tra alternative di selezione aperte
  - Scelta tra operazioni possibili entro una risorsa protetta

Corso di Laurea Magistrale in Informatica, Università di Padova 2/17



Correttezza temporale

## Premesse – 2

- ❑ I sistemi a tempo reale devono assicurare correttezza temporale oltre che funzionale
- ❑ Devono poter esercitare controllo sul proprio grado di non-determinismo
  - Implicitamente : per selezione di politiche di accodamento e/o selezione e assegnazione di attributi (configurazione)
  - Esplicitamente : in forma algoritmica

Corso di Laurea Magistrale in Informatica, Università di Padova 3/17



Correttezza temporale

## Politiche di ordinamento – 1

- ❑ FPP : *fixed priority preemptive*
  - Originariamente la sola opzione disponibile
    - `pragma Task_Dispatching_Policy (FIFO_Within_Priorities)`
    - *Thread* con priorità uguale sono gestiti con accodamento FIFO
  - Nel seguito assumeremo questa politica ←
- ❑ FPNP : come sopra ma senza prerilascio
  - L'arrivo di un *thread* a priorità maggiore non comporta prerilascio
  - Il rilascio è volontario (cioè cooperativo) tramite invocazione esplicita di "`delay 0.0`" (meglio: `yield`)
    - `pragma Task_Dispatching_Policy (Non_Preemptive_FIFO_Within_Priorities)`

Corso di Laurea Magistrale in Informatica, Università di Padova 4/17



Correttezza temporale

## Politiche di ordinamento – 2

- ❑ RR : *round robin*
  - Ampiezza del quanto predefinita o fissata da `Ada.Dispatching.Round_Robin.Set_Quantum (...)`
    - `pragma Task_Dispatching_Policy (Round_Robin_Within_Priorities)`
- ❑ EDF : *earliest deadline first*
  - I *thread* hanno un attributo "scadenza" (*deadline*) che ne determina l'urgenza
    - Scadenza relativa iniziale (tramite `pragma Relative_Deadline (...)`) e poi assoluta via `Ada.Dispatching.EDF.Set_Deadline (...)`
    - `pragma Task_Dispatching_Policy (EDF_Across_Priorities)`

Corso di Laurea Magistrale in Informatica, Università di Padova 5/17



Correttezza temporale

## Priorità d'esecuzione – 1

- ❑ I processi hanno un attributo predefinito con effetto sull'ordinamento
  - Priorità di base
- ❑ Assegnabile tramite comando di configurazione `pragma Priority(N)`
  - Per *N* valore intero in un intervallo fissato per piattaforma
  - Se non configurata esplicitamente la priorità viene assunta essere uguale alla priorità di base del processo padre
  - L'unità *main* viene vista come un processo implicito con priorità
    - Dunque possibile "padre" di processi oltre che loro "master"

Corso di Laurea Magistrale in Informatica, Università di Padova 6/17

Correttezza temporale

## Priorità d'esecuzione – 2

- ❑ La mutua esclusione in accesso a risorse protette può confliggere con le politiche di ordinamento
- ❑ Un regime di ordinamento a priorità si impegna a garantire che, a ogni istante, il processo in esecuzione sia **sempre** quello a priorità maggiore
- ❑ Se ciò non accade la situazione viene detta di "inversione di priorità"
  - Rischio di violazione della proprietà di correttezza temporale

Corso di Laurea Magistrale in Informatica, Università di Padova 7/17

Correttezza temporale

## Priorità d'esecuzione – 3

Corso di Laurea Magistrale in Informatica, Università di Padova 8/17

Correttezza temporale

## Priorità d'esecuzione – 4

- ❑ L' inversione di priorità che ritarda il processo H ha 2 componenti distinte
  - Il blocco della risorsa condivisa P da parte di L, meno importante di H
    - Durata irriducibile (intrinseca del modello di sincronizzazione)
  - L'esecuzione di M, meno importante di H, ma più importante di L, che **ritarda** il rilascio di P a discapito di H
    - Durata riducibile
- ❑ Il modello deve minimizzare la durata riducibile
  - Varie tecniche consentono di farlo con diversa efficacia
  - Tutte ispirate all'ereditarietà della priorità maggiore
  - Esercizio 12  
Mostrare come questa tecnica risolve il problema dell'esempio

Corso di Laurea Magistrale in Informatica, Università di Padova 9/17

Correttezza temporale

## Priorità d'esecuzione – 5

- ❑ **Basic priority inheritance protocol**
  - La priorità di un *thread* varia nel tempo rispetto a quella assegnata inizialmente dall'algoritmo di *scheduling*
  - La variazione avviene per ereditarietà
- ❑ **Regole del protocollo**
  - Quando un *thread* J richiede una risorsa R al tempo t
    - Se R è libera, R viene assegnata a J fino al suo rilascio
    - Se R è occupata, la richiesta viene negata e J diventa bloccato
  - Quando J diventa bloccato, il *thread* J, che lo blocca assume la priorità di J e la detiene fino al rilascio di R quando J<sub>i</sub> riassume la sua priorità precedente

Corso di Laurea Magistrale in Informatica, Università di Padova 10/17

Correttezza temporale

## Priorità d'esecuzione – 6

- ❑ **Immediate priority ceiling inheritance protocol**
  - Come in BPI
  - In più ogni RP ha una priorità statica, detta *ceiling*, non inferiore alla priorità maggiore fra quelle dei suoi processi cliente
- ❑ **Regole del protocollo**
  - Quando un *thread* esegue all'interno di una RP esso assume **immediatamente** la priorità della risorsa per tutta (e sola) la durata dell'esecuzione protetta
  - In pratica, quando un *thread* J richiede una risorsa R, essa gli viene concessa
- ❑ Questa politica **azzer**a la durata riducibile del periodo di inversione di priorità

Corso di Laurea Magistrale in Informatica, Università di Padova 11/17

Correttezza temporale

## Priorità d'esecuzione – 7

- ❑ In ambiente *single-core* IPCI è sufficiente a garantire mutua esclusione
  - Un processo che opera entro una RP esegue in preferenza a tutti gli altri processi cliente e anche a tutti i processi "non cliente" a priorità inferiore al *ceiling* della risorsa
  - Priorità della risorsa **strettamente maggiore** di quella dei suoi processi clienti
    - Garanzia assoluta di mutua esclusione
  - Priorità della risorsa **uguale alla maggiore** tra quelle dei suoi clienti
    - Garanzia di mutua esclusione solo in assenza di prerilascio tra processi a pari priorità (modalità *round robin*)

Corso di Laurea Magistrale in Informatica, Università di Padova 12/17

Correttezza temporale

Priorità d'esecuzione – 8

❑ IPCI ha altre 2 proprietà importanti in ambiente monoprocesso

- Ogni processo subisce al più 1 ritardo (blocco) da inversione di priorità irriducibile e solo al suo rilascio
- Se tutte le risorse condivise sono accedute sotto regime IPCI e le loro priorità sono coerentemente assegnate allora non si può verificare stallo
  - Esercizio 13: Individuare le precondizioni di stallo impeditte da ICPI su *single-core*

❑ Il programma diventa erroneo se un processo tenta di accedere un risorsa avendo priorità superiore a esso (*ceiling violation*)

Corso di Laurea Magistrale in Informatica, Università di Padova
13/17

Correttezza temporale

Priorità d'esecuzione – 9

❑ La realizzazione di IPCI si presta a una interessante ottimizzazione

- Il processo in uscita da una RP può eseguire anche le richieste pendenti in code con guardia aperta per conto dei relativi processi cliente
  - *Proxy model*

❑ Questa ottimizzazione

- Preserva l'esecuzione in mutua esclusione nella risorsa
- Riduce l'onere di cambio di contesto tra processi clienti

Corso di Laurea Magistrale in Informatica, Università di Padova
14/17

Correttezza temporale

Proxy model: esempio

```
protected Gate_Control is
pragma Priority (28);
entry Stop_And_Close;
procedure Open;
private
Gate : Boolean := False;
end Gate_Control;
protected body Gate_Control is
entry Stop_And_Close when Gate is
begin
Gate := False;
end Stop_And_Close;
procedure Open is
begin
Gate := True;
end Open;
end Gate_Control;
```

①

T  
Priority (20)

②

S  
Priority (27)

T si accoda su (1) attualmente chiusa  
S esegue (2) e apre (1)  
T può procedere  
S può eseguire le azioni richieste da T al suo posto, risparmiando 2 scambi di contesto con esso

Perché 2?

Corso di Laurea Magistrale in Informatica, Università di Padova
15/17

Correttezza temporale

Priorità d'esecuzione – 10

❑ L' ereditarietà di priorità comporta che ogni processo abbia 2 attributi di priorità

- Priorità di base → assegnata alla definizione del processo
- Priorità attiva → a fini di ordinamento,  $\max\{PB, PE\}$

❑ Si ha ereditarietà di priorità

- All'accesso in risorsa protetta
- Durante l'attivazione di un processo figlio a priorità maggiore, quando il padre ne assume la priorità per limitare il suo tempo di blocco
- Durante un *rendezvous*, quando il servente assume la priorità del cliente (se >) per la durata della sincronizzazione
  - Ma il servente esegue alla sua propria priorità fuori dalla sincronizzazione

Corso di Laurea Magistrale in Informatica, Università di Padova
16/17

Correttezza temporale

Politiche di accodamento

❑ Coda dei processi pronti → politica di ordinamento

- A priorità maggiore e FIFO tra priorità uguali (*FIFO\_Within\_Priorities*)
  - Ogni processo che diventa pronto viene posto in fondo alla coda tra i processi pronti alla sua stessa priorità
  - Un processo in esecuzione scalzato da prerilascio viene posto in testa alla coda dei processi pronti alla sua stessa priorità

❑ Coda su canale tipato con guardia

- FIFO all'interno della stessa coda
- A priorità attiva tra tutte le chiamate in tutte le code dell'entità (servente o risorsa protetta) con guardia aperta
  - Tramite `pragma Queuing_Policy (...)` con argomento `FIFO_Queueing` o `Priority_Queueing`

Corso di Laurea Magistrale in Informatica, Università di Padova
17/17