Correttezza temporale

SCD

Anno accademico 2008/9
Corso di Sistemi Concorrenti e Distribuiti

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

Premesse – 1

- La correttezza funzionale di un programma concorrente non dovrebbe dipendere dall'ordine d'esecuzione dei processi
- Può essere necessario dimostrare che il non-determinismo dell'esecuzione non incorra in stallo (*deadlock*) o avanzamento senza progresso (*livelock*)
- Il modello di concorrenza visto a lezione 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

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

Politiche di ordinamento

- 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 (cooperativo) tramite invocazione di "delay 0.0"
    - **pragma** Task_Dispatching_Policy (Non_Preemptive_FIFO_Within_Priorities)
- RR : *round robin*
  - Quanto predefinito o fissato 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)

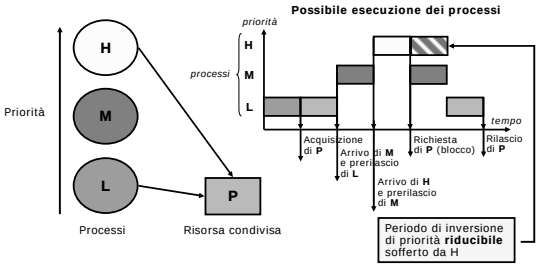
Priorità d'esecuzione – 1

- 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
    - Dunque possibile "padre" di processi oltre che loro "master"

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

Priorità d'esecuzione – 3



Priorità d'esecuzione – 4

- **Il processo H è ritardato da 2 componenti di inversione di priorità**
  - **Il blocco della risorsa condivisa P da parte di L, meno importante di H**
    - **Durata irriducibile** (intrinseca del modello)
  - **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 essere capace di minimizzare la durata riducibile**
  - **Varie tecniche consentono di farlo con diversa efficacia**
  - **Tutte ispirate all'ereditarietà della priorità maggiore**
  - **Esercizio 13**
    Mostrare come questa tecnica risolva il problema dell'esempio

Priorità d'esecuzione – 5

- **Ereditarietà immediata della priorità più elevata**
  - ***Immediate priority ceiling (inheritance) protocol → ICPI***
  - **A ogni RP viene attribuita una priorità non inferiore alla priorità maggiore fra quelle dei suoi processi cliente**
    - *Priority ceiling*
  - **Ogni volta che un processo cliente esegue all'interno di una RP esso assume immediatamente la priorità della risorsa per tutta (e sola) la durata dell'esecuzione protetta**
- **Questa politica azzera la durata riducibile del periodo di inversione di priorità**

Priorità d'esecuzione – 6

- **In ambiente mono-processore ICPI è sufficiente a garantire mutua esclusione**
  - **Quando un processo opera entro una RP esso 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 assoluta di mutua esclusione
    - Ma solo in assenza di prerilascio tra processi a pari priorità (modalità *round robin*)

Priorità d'esecuzione – 7

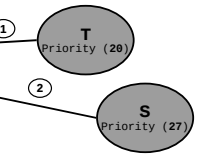
- **ICPI ha altre 2 proprietà importanti in ambiente monoprocessore**
  - **Ogni processo subisce al più 1 ritardo (blocco) da inversione di priorità irriducibile e solo al suo rilascio**
  - **Se tutte le risorse condivise sono accedute sotto regime ICPI e le loro priorità sono coerentemente assegnate allora non si può verificare stallo**
    - **Esercizio 14**: Individuare le precondizioni di stallo impedite da ICPI su monoprocessore
- **Il programma diventa erroneo se un processo tenta di accedere un risorsa avendo priorità superiore a esso (*ceiling violation*)**

Priorità d'esecuzione – 8

- **La realizzazione di ICPI 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**

Priorità d'esecuzione – 9

```
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;
```



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?

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 il padre ne assume la priorità per limitare il suo tempo di blocco**
  - **Durante un *rendezvous* il servente assume la priorità del cliente (se >) per la durata della sincronizzazione**
    - Ma il servente esegue alla sua propria priorità fuori dalla sincronizzazione

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**
    - Soddisfacendo il requisito 4 di Toby Bloom (lezione C07, pagina 3)
    - Tramite **pragma** Queuing_Policy (...) con argomento FIFO_Queuing o Priority_Queuing