Correttezza temporale

SCD

Anno accademico 2012/13
Sistemi Concorrenti e Distribuiti

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

Premesse – 1

- La correttezza funzionale di un programma concorrente non deve dipendere dall'ordine d'esecuzione dei processi
- Il non-determinismo dell'esecuzione non deve causare *deadlock* o *livelock*
- Il modello di concorrenza visto finora 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 – 1

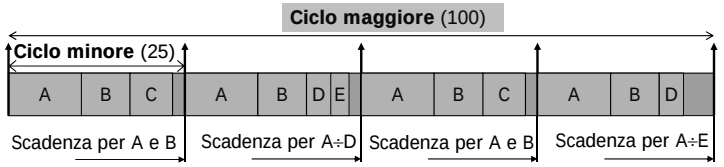
- FPP : *fixed priority preemptive*
  - Originariamente la sola opzione disponibile (*default*)
    - `pragma Task_Dispatching_Policy (FIFO_Within_Priorities)`
    - *Thread* con priorità uguale sono gestiti con accodamento FIFO
- FPNP : come sopra ma senza prerilascio
  - L'arrivo di un *thread* a priorità maggiore non causa prerilascio immediato
  - Il rilascio è volontario (cooperativo) tramite invocazione esplicita di `yield` (oppure `delay 0.0`)
    - `pragma Task_Dispatching_Policy (Non_Preemptive_FIFO_Within_Priorities)`

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***
  - ***Thread* con attributo "deadline" che ne fissa 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)`

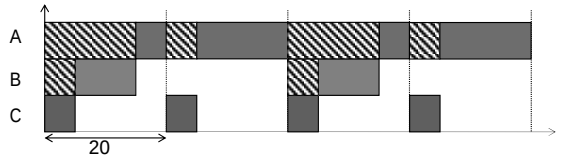
Ordinamento a quanti ≠ RR

Processo	Periodo	Durata
A	25	10
B	25	8
C	50	5
D	50	4
E	100	2

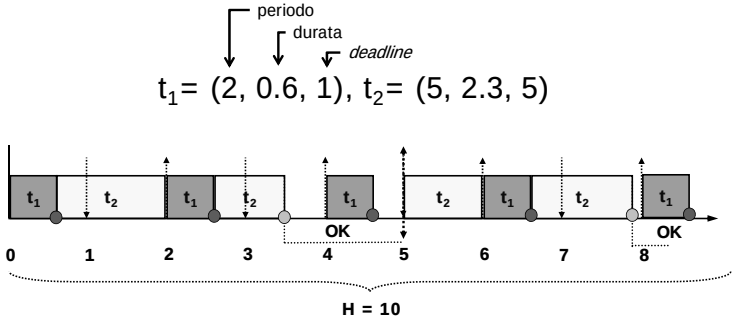


Ordinamento FPS

Processo	Periodo	Durata	Priorità
A	80	40	1 (L)
B	40	10	2
C	20	5	3 (H)



Ordinamento EDF



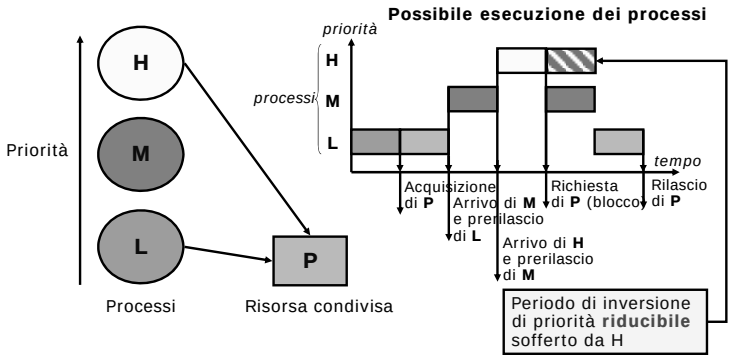
Priorità d'esecuzione – 1

- **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 fissata esplicitamente viene assunta uguale alla priorità di base del processo padre**
  - **Il *main* è visto come un processo implicito con priorità**
    - Possibile «padre» di processi «figli» 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

- **L' inversione di priorità che ritarda il processo H ha 2 componenti distinte**
  - **Il blocco di P da parte di L meno importante di H**
    - Durata irriducibile → intrinseca nella 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 → causata dall'interazione tra il protocollo di sincronizzazione e le politiche di ordinamento
- **Il modello deve minimizzare la durata riducibile**
  - **Varie tecniche consentono di farlo con diversa efficacia**
  - **Tutte ispirate all'ereditarietà della priorità maggiore**

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_l che lo blocca assume la priorità di J e la detiene fino al rilascio di R quando J_l riassume la sua priorità precedente**

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 azzera la durata riducibile del periodo di inversione di priorità**

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*)

Priorità d'esecuzione – 8

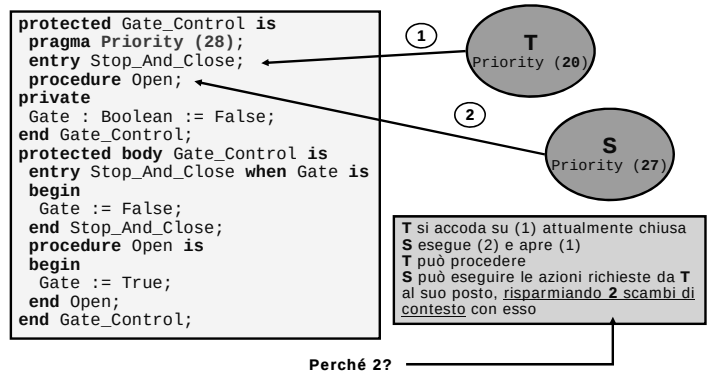
- **IPCI ha altre 2 proprietà importanti in ambiente *single-core***
  - **Ogni processo subisce al più 1 ritardo («blocco») da inversione di priorità irriducibile e solo al suo rilascio**
  - **Se tutte le risorse condivise sono accedute sotto regime IPCI e le loro priorità sono coerentemente assegnate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

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**

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;
```



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 *rendez-vous*, 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

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_Queuing` o `Priority_Queuing`