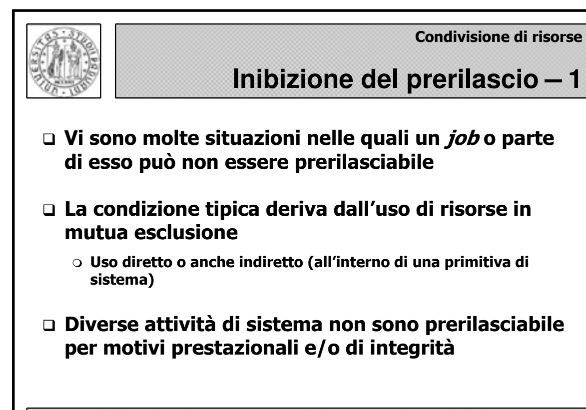
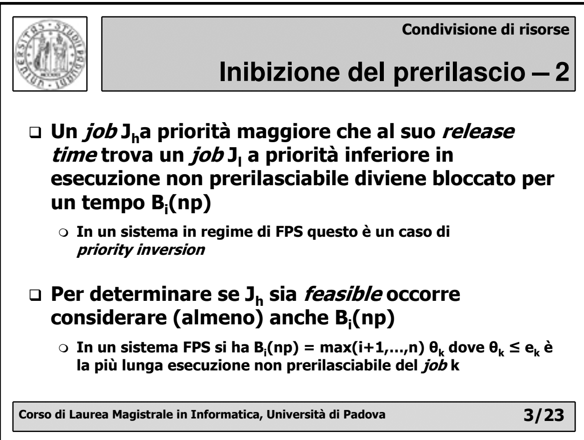
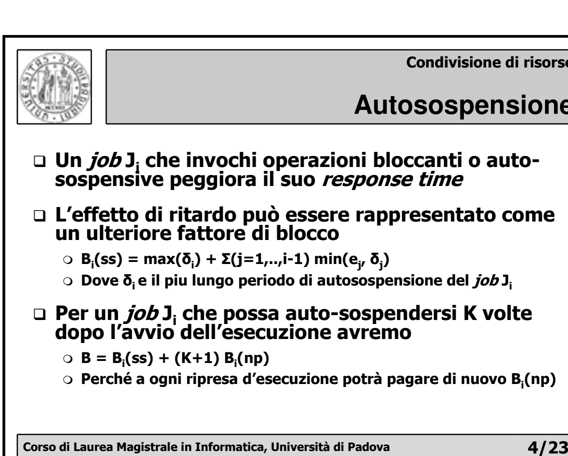
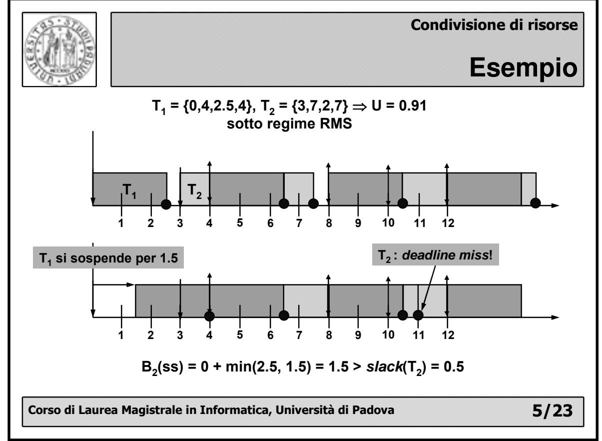
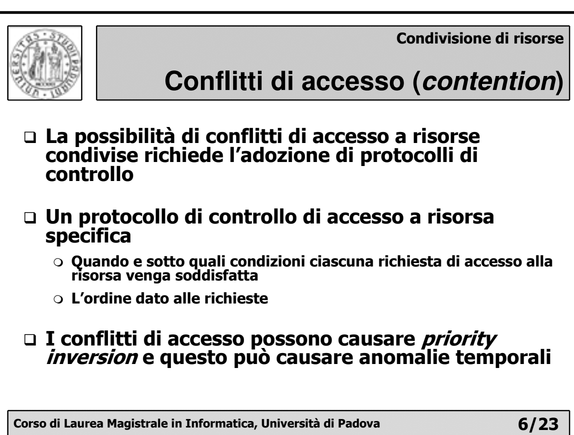
Condivisione di risorse
RTS
Anno accademico 2008/9
Sistemi Real-Time
Tullio Vardanega, tullio.vardanega@math.unipd.it
Corso di Laurea Magistrale in Informatica, Università di Padova
1/23

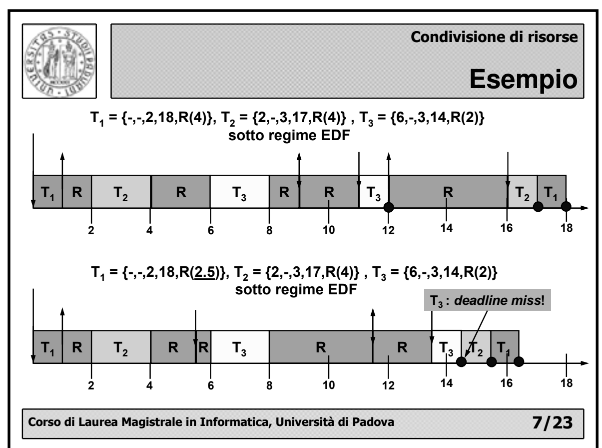

Condivisione di risorse
Inibizione del prerilascio – 1
Vi sono molte situazioni nelle quali un job o parte di esso può non essere prerilasciabile
La condizione tipica deriva dall'uso di risorse in mutua esclusione
Uso diretto o anche indiretto (all'interno di una primitiva di sistema)
Diverse attività di sistema non sono prerilasciabile per motivi prestazionali e/o di integrità
Corso di Laurea Magistrale in Informatica, Università di Padova
2/23

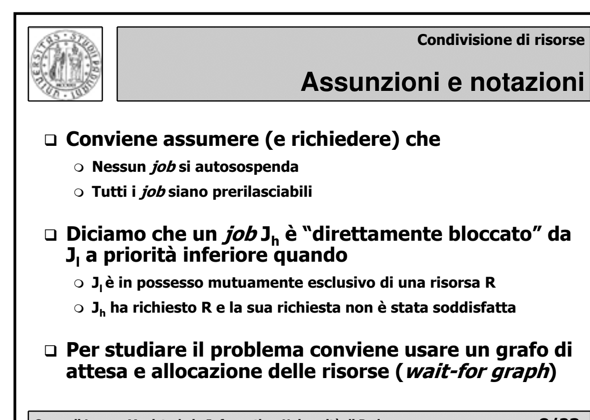

Condivisione di risorse
Inibizione del prerilascio – 2
Un job J_h a priorità maggiore che al suo release time trova un job J_l a priorità inferiore in esecuzione non prerilasciabile diviene bloccato per un tempo $B_i(np)$
In un sistema in regime di FPS questo è un caso di priority inversion
Per determinare se J_h sia feasible occorre considerare (almeno) anche $B_i(np)$
In un sistema FPS si ha $B_i(np) = max(i+1,\dots,n)\ \theta_k$ dove $\theta_k \le e_k$ è la più lunga esecuzione non prerilasciabile del job k
Corso di Laurea Magistrale in Informatica, Università di Padova
3/23

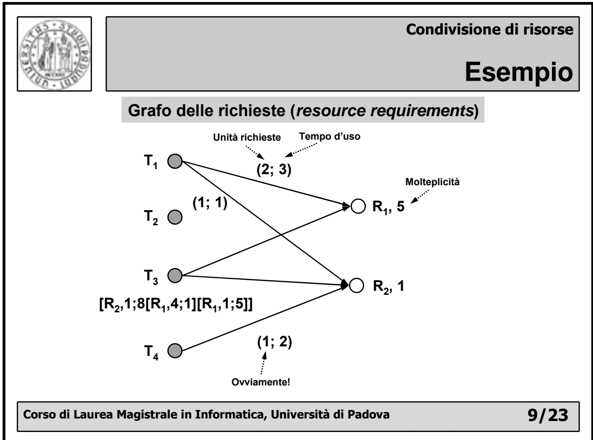

Condivisione di risorse
Autosospensione
Un job J_i che invochi operazioni bloccanti o auto-sospensive peggiora il suo response time
L'effetto di ritardo può essere rappresentato come un ulteriore fattore di blocco
$B_i(ss) = max(\delta_i) + \Sigma(j=1,..,i-1)\ min(e_j, \delta_j)$
Dove δ_i e il piu lungo periodo di autosospensione del job J_i
Per un job J_i che possa auto-sospendersi K volte dopo l'avvio dell'esecuzione avremo
$B = B_i(ss) + (K+1)\ B_i(np)$
Perché a ogni ripresa d'esecuzione potrà pagare di nuovo $B_i(np)$
Corso di Laurea Magistrale in Informatica, Università di Padova
4/23

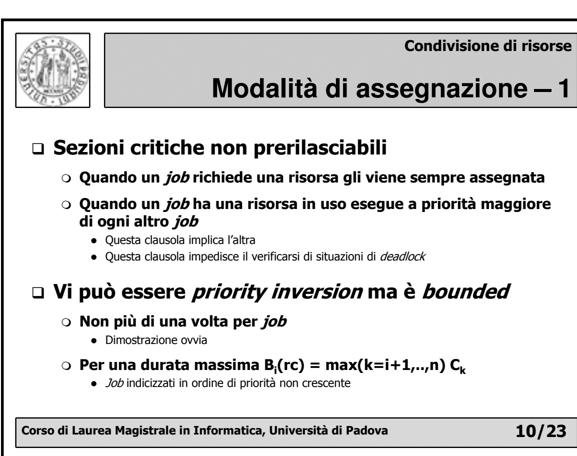

Condivisione di risorse
Esempio
$T_1 = \{0,4,2.5,4\}$, $T_2 = \{3,7,2,7\} \Rightarrow U = 0.91$
sotto regime RMS
T_1
T_2
1 2 3 4 5 6 7 8 9 10 11 12
T_1 si sospende per 1.5
T_2 : deadline miss!
1 2 3 4 5 6 7 8 9 10 11 12
$B_2(ss) = 0 + min(2.5, 1.5) = 1.5 > slack(T_2) = 0.5$
Corso di Laurea Magistrale in Informatica, Università di Padova
5/23

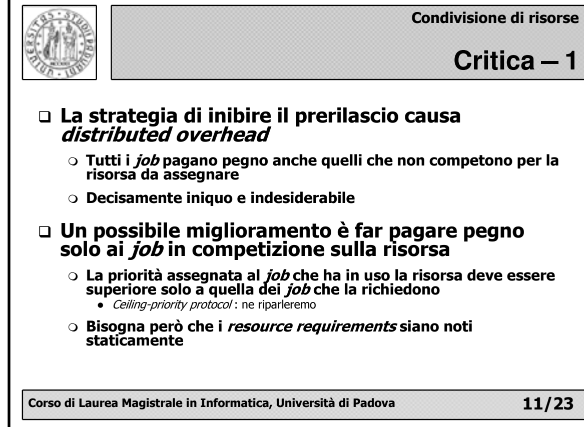

Condivisione di risorse
Conflitti di accesso (contention)
La possibilità di conflitti di accesso a risorse condivise richiede l'adozione di protocolli di controllo
Un protocollo di controllo di accesso a risorsa specifica
Quando e sotto quali condizioni ciascuna richiesta di accesso alla risorsa venga soddisfatta
L'ordine dato alle richieste
I conflitti di accesso possono causare priority inversion e questo può causare anomalie temporali
Corso di Laurea Magistrale in Informatica, Università di Padova
6/23

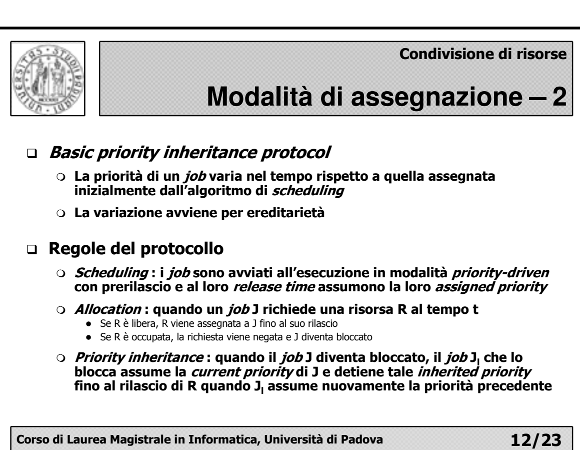

Condivisione di risorse
Esempio
T1 = {-,-,2,18,R(4)}, T2 = {2,-,3,17,R(4)} , T3 = {6,-,3,14,R(2)}
sotto regime EDF
T1 R T2 R T3 R R T3 R T2 T1
2 4 6 8 10 12 14 16 18
T1 = {-,-,2,18,R(2.5)}, T2 = {2,-,3,17,R(4)} , T3 = {6,-,3,14,R(2)}
sotto regime EDF
T3 : deadline miss!
T1 R T2 R R T3 R R T3 T2 T1
2 4 6 8 10 12 14 16 18
Corso di Laurea Magistrale in Informatica, Università di Padova
7/23


Condivisione di risorse
Assunzioni e notazioni
Conviene assumere (e richiedere) che
Nessun job si autosospenda
Tutti i job siano prerilasciabili
Diciamo che un job Jh è "direttamente bloccato" da Jl a priorità inferiore quando
Jl è in possesso mutuamente esclusivo di una risorsa R
Jh ha richiesto R e la sua richiesta non è stata soddisfatta
Per studiare il problema conviene usare un grafo di attesa e allocazione delle risorse (wait-for graph)
Corso di Laurea Magistrale in Informatica, Università di Padova
8/23


Condivisione di risorse
Esempio
Grafo delle richieste (resource requirements)
Unità richieste
Tempo d'uso
T1
(2; 3)
Molteplicità
(1; 1)
R1, 5
T2
T3
R2, 1
[R2,1;8[R1,4;1][R1,1;5]]
T4
(1; 2)
Ovviamente!
Corso di Laurea Magistrale in Informatica, Università di Padova
9/23


Condivisione di risorse
Modalità di assegnazione – 1
Sezioni critiche non prerilasciabili
Quando un job richiede una risorsa gli viene sempre assegnata
Quando un job ha una risorsa in uso esegue a priorità maggiore di ogni altro job
Questa clausola implica l'altra
Questa clausola impedisce il verificarsi di situazioni di deadlock
Vi può essere priority inversion ma è bounded
Non più di una volta per job
Dimostrazione ovvia
Per una durata massima Bi(rc) = max(k=i+1,..,n) Ck
Job indicizzati in ordine di priorità non crescente
Corso di Laurea Magistrale in Informatica, Università di Padova
10/23


Condivisione di risorse
Critica – 1
La strategia di inibire il prerilascio causa distributed overhead
Tutti i job pagano pegno anche quelli che non competono per la risorsa da assegnare
Decisamente iniquo e indesiderabile
Un possibile miglioramento è far pagare pegno solo ai job in competizione sulla risorsa
La priorità assegnata al job che ha in uso la risorsa deve essere superiore solo a quella dei job che la richiedono
Ceiling-priority protocol : ne riparleremo
Bisogna però che i resource requirements siano noti staticamente
Corso di Laurea Magistrale in Informatica, Università di Padova
11/23


Condivisione di risorse
Modalità di assegnazione – 2
Basic priority inheritance protocol
La priorità di un job varia nel tempo rispetto a quella assegnata inizialmente dall'algoritmo di scheduling
La variazione avviene per ereditarietà
Regole del protocollo
Scheduling : i job sono avviati all'esecuzione in modalità priority-driven con prerilascio e al loro release time assumono la loro assigned priority
Allocation : quando un job 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
Priority inheritance : quando il job J diventa bloccato, il job Jl che lo blocca assume la current priority di J e detiene tale inherited priority fino al rilascio di R quando Jl assume nuovamente la priorità precedente
Corso di Laurea Magistrale in Informatica, Università di Padova
12/23

Condivisione di risorse

Critica – 2

- BPIP comporta due forme di *blocking*
  - *Direct blocking* : dovuto alla competizione
  - *Inheritance blocking* : dovuto all'innalzamento della priorità
- L'ereditarietà è transitiva perché il *direct blocking* è reso transitivo dalla possibilità per i *job* di cumulare risorse
- BPIP non evita il *deadlock* perché permettendo il blocco transitivo consente anche quello ciclico
- Causa *distributed overhead* riducibile
  - Sotto BPIP un *job* può essere bloccato più volte quando competa con più *job* per più di una risorsa

Condivisione di risorse

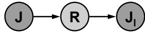
Modalità di assegnazione – 3

- *Basic priority ceiling protocol*
  - Come sotto BPIP, e in più con *resource requirements* noti staticamente
  - Ogni risorsa R ha *priority ceiling* fissato alla priorità maggiore fra quelle dei *job* che richiedono R
    - All'instante t il sistema ha un *ceiling* Π(t) dato dal *priority ceiling* maggiore tra tutte le risorse in uso al tempo t oppure Ω, valore fittizio, inferiore alla minima priorità dei *job*
- Regole del protocollo
  - *Scheduling* : i *job* sono avviati all'esecuzione in modalità *priority-driven* con prerilascio e al loro *release time* assumono la loro *assigned priority*
  - *Allocation* : quando un job J richiede una risorsa R al tempo t
    - R è in uso a un altro *job*, la richiesta di J fallisce e J diventa bloccato
    - R è libera e la priorità di J è > Π(t), R viene assegnata a J
    - Altrimenti, se J possiede la risorsa con *priority ceiling* = Π(t), R viene assegnata a J, diversamente la richiesta di J fallisce a J diventa bloccato
  - *Priority inheritance* : quando J diventa bloccato, il *job* J_l che lo blocca assume la *current priority* π(t) di J ed esegue fino al rilascio di tutte le sue risorse con *priority ceiling* ≥ π(t), quando la priorità di J_l ritorna al valore precedente alla loro acquisizione

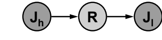
Condivisione di risorse

Critica – 3

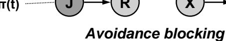
- BPIP è *greedy* dove BPCP non lo è
  - In BPCP una richiesta può fallire anche a risorsa libera
- Sotto BPCP ogni job J è esposto a 3 forme di *blocking* causate da J_l



Direct blocking



Priority-inheritance blocking



Avoidance blocking

Condivisione di risorse

Critica – 4

- L'*avoidance blocking* che rende BPCP non *greedy* impedisce il formarsi di *deadlock*
  - Che al tempo t per un *job* J con *current priority* π(t) valga π(t) > Π(t) significa che
    - J non userà mai alcuna delle risorse occupate al tempo t
    - I *job* con priorità ≥ di J neppure
  - Il valore del *ceiling* Π(t) determina il sottoinsieme di *job* cui si possano assegnare risorse libere al tempo t senza rischio di *deadlock*
    - Tutti i *job* con priorità maggiore del *ceiling* Π(t)
- Caveat
  - Per evitare che J blocchi se stesso quando ha bisogno di cumulare risorse la risorsa R gli deve essere assegnata se π(t) ≤ Π(t) ma J possiede le risorse {X} con *priority ceiling* = Π(t)

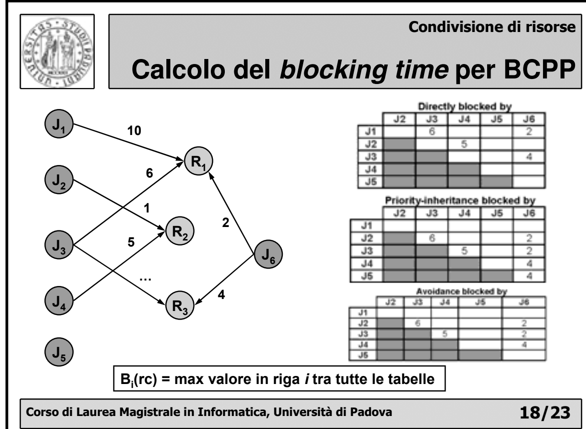
Condivisione di risorse

Critica – 5

- BPCP non causa *distributed overhead* riducibile perché non permette *transitive blocking*
- Teorema [Sha & Rajkumar & Lehoczky, 1990]: sotto BPCP un *job* può essere bloccato al più per la durata di una sezione critica
  - Quando un *job* diventa bloccato sotto BPCP questo può essere solo a causa di un solo *job*
  - Un *job* che causa blocco non può subirne (i.e., il blocco non è transitivo)
- Il valore massimo di tale durata è detto *blocking time* causato da conflitto di accesso a risorse
  - $B_i(rc)$ deve essere considerato nello *schedulability test* di J_i

Condivisione di risorse

Calcolo del *blocking time* per BCPP



Directly blocked by

	J2	J3	J4	J5	J6
J1		6			2
J2			5		
J3					4
J4					
J5					

Priority-inheritance blocked by

	J2	J3	J4	J5	J6
J1					
J2		6			2
J3			5		2
J4					4
J5					4

Avoidance blocked by

	J2	J3	J4	J5	J6
J1					
J2		6			2
J3			5		2
J4					4
J5					

$B_i(rc)$ = max valore in riga *i* tra tutte le tabelle

Condivisione di risorse

Annotazioni sul calcolo

- **Tabella "*priority-inheritance blocked by*"**
  - **Il valore della cella [i,k] è dato dal massimo valore tra quelli presenti nella righe 1,…,i-1 della colonna k della tabella "*directly blocked by*"**
- **Tabella "*avoidance blocked by*"**
  - **Nel caso (raccomandabile) che tutti i *job* abbiano priorità distinte, le celle di questa tabella sono uguali alle corrispondenti celle della tabella "*priority-inheritance blocked by*" eccetto per i *job* che non richiedono alcuna risorsa (per i quali il valore è nullo)**

Corso di Laurea Magistrale in Informatica, Università di Padova 19/23

Condivisione di risorse

Modalità di assegnazione – 4

- ***(Stack-based) ceiling priority protocol***
  - **Si può far meglio di BPCP**
  - **Per risparmio di risorse preziose (condivisione di *stack* tra tutti i *job*)**
    - Per evitare che un *job* abbia uno spazio di *stack* frammentato – a causa del prerilascio – bisogna garantire che non gli venga negata alcuna risorsa durante la sua esecuzione
    - Ovviamente gli deve essere anche impedita l'auto-sospensione!
  - **Per riduzione di complessità algoritmica**
    - Non conviene sovraccaricare il *run-time support* in termini di spazio (strutture dati di controllo) e di tempo (esecuzione di protocolli)

Corso di Laurea Magistrale in Informatica, Università di Padova 20/23

Condivisione di risorse

Ceiling priority protocol – 1

- **Versione *stack-based* [Baker, 1991]**
  - **Calcolo e gestione del *ceiling* $\Pi(t)$: Quando tutte le risorse sono libere $\Pi(t)$ vale Ω e il suo valore viene aggiornato ogni volta che una risorsa viene assegnata o rilasciata**
  - ***Scheduling* : al suo *release time* un *job* resta bloccato fino a quando la sua *assigned priority* $\pi(t)$ non sia maggiore di $\Pi(t)$**
    - A ogni instante i *job* non bloccati sono sono avviati all'esecuzione in modalità *priority-driven* con prerilascio
  - ***Allocation* : quando un *job* richiede una risorsa questa gli viene assegnata**

Corso di Laurea Magistrale in Informatica, Università di Padova 21/23

Condivisione di risorse

Osservazioni

- **Sotto SB-CPP un *job* inizia a eseguire solo quando le risorse di cui necessita sono libere**
  - **Altrimenti non potrebbe essere che $\pi(t) > \Pi(t)$**
- **Sotto SB-CPP un *job* che venga pre-rilasciato non resta bloccato**
  - **Perché il *job* che prerilascia non può richiedere alcuna delle risorse necessarie al *job* prerilasciato**
- **Sotto SB-CPP non si può verificare *deadlock***
- **Il valore di $B_i(rc)$ sotto SB-CPP equivale a quello risultante da BPCP**

Corso di Laurea Magistrale in Informatica, Università di Padova 22/23

Condivisione di risorse

Ceiling priority protocol – 2

- **Versione *ceiling priority protocol***
  - **Non si usa il *ceiling* $\Pi(t)$ ma le risorse continuano ad avere *ceiling priority***
  - ***Scheduling***
    - Ogni *job* esegue al livello della sua *assigned priority* quando non detiene alcuna risorsa
    - *Job* con la stessa priorità sono ordinati su base FIFO (`FIFO_within_priorities`)
    - La priorità di un *job* che detenga risorse assume il valore massimo tra i *ceiling* di tali risorse
  - ***Allocation* : quando un *job* richiede una risorsa questa gli viene assegnata**

Corso di Laurea Magistrale in Informatica, Università di Padova 23/23