

Lezione R10

Controllo d'accesso alle risorse condivise – II

Sistemi embedded e real-time

29 ottobre 2020

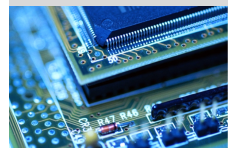
Marco Cesati

Dipartimento di Ingegneria Civile e Ingegneria Informatica
Università degli Studi di Roma Tor Vergata

Di cosa parliamo in questa lezione?

In questa lezione terminiamo il discorso sui protocolli di controllo d'accesso alle risorse condivise

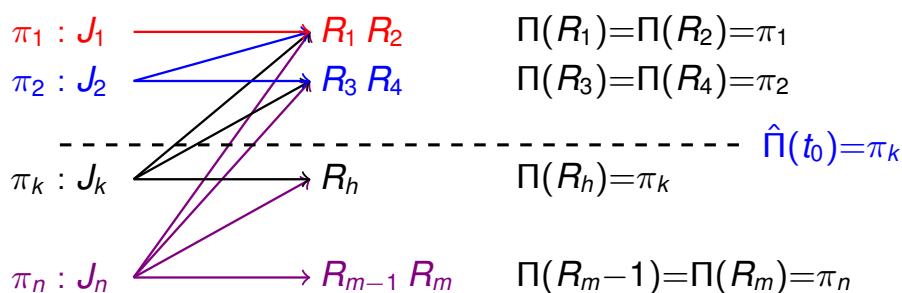
- 1 Proprietà del protocollo priority-ceiling
- 2 Protocollo stack-based priority-ceiling
- 3 Protocollo ceiling-priority
- 4 Gestire job con auto-sospensione
- 5 Protocolli per priorità dinamica
- 6 Accesso alle risorse di job aperiodici



Il protocollo priority-ceiling

- Assegna ad ogni risorsa il valore **priority-ceiling** che indica la massima priorità tra i job che usano la risorsa
- Mantiene aggiornato il **current priority ceiling** del sistema $\hat{\Pi}(t)$ che indica il massimo valore associato a tutte le risorse assegnate
- Job bloccanti ereditano la priorità dinamica dei job bloccati
- Al tempo t un solo job possiede tutte le risorse assegnate aventi **priority ceiling** uguale a $\hat{\Pi}(t)$
- Se un job sta per ottenere una risorsa e $\pi(t) > \hat{\Pi}(t)$, nessun job di priorità uguale o superiore (compreso il job stesso) ha richiesto o richiederà risorse già assegnate
- Se un job sta per ottenere una risorsa e $\pi(t) = \hat{\Pi}(t)$, il job è il possessore di tutte le risorse assegnate aventi **priority ceiling** uguale a $\hat{\Pi}(t)$
- I deadlock sono evitati

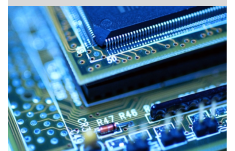
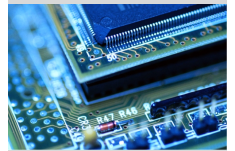
Come si evitano i deadlock nel protocollo priority-ceiling



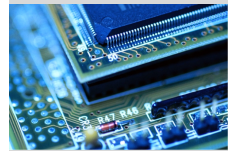
Se al tempo t_0 un job J richiede una risorsa R e $\pi(t_0) > \hat{\Pi}(t_0)$:

- J non chiederà mai alcuna risorsa già assegnata al tempo t_0
 \Rightarrow nessun deadlock con risorse già assegnate
- Nessun job con priorità $\geq \pi(t_0)$ chiederà alcuna risorsa già assegnata al tempo t_0
 \Rightarrow nessun job che già possiede una risorsa al tempo t_0 potrà interrompere J e richiedere R

\Rightarrow Il protocollo **priority-ceiling** evita i deadlock



Durata dei blocchi nel protocollo priority-ceiling



Tre possibili cause di blocco: blocco **diretto**, per **priority-inheritance**, e per **priority-ceiling**

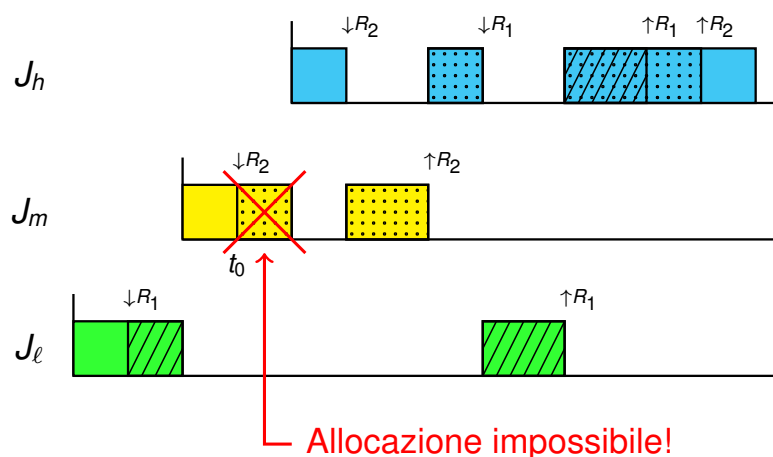
Teorema

Utilizzando il protocollo priority-ceiling un job può essere bloccato al massimo per la durata di **una** sezione critica

Il teorema è conseguenza di due proprietà:

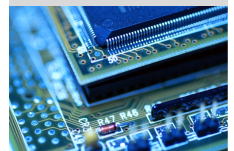
- 1 Se un job viene bloccato, è bloccato da un solo job
- 2 Non esiste **blocco transitivo**: non si verifica mai il caso J_3 blocca J_2 e J_2 blocca J_1

Unicità del job bloccante

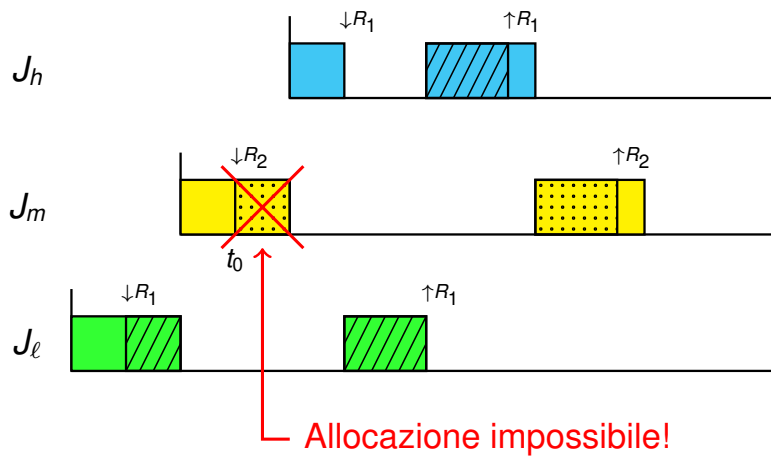


- $\pi_h > \pi_m > \pi_\ell \Rightarrow \Pi(R_1) \geq \pi_h$ e $\Pi(R_2) \geq \pi_h$
- $\hat{\Pi}(t_0) \geq \Pi(R_1) \geq \pi_h$
- Requisito per allocazione a t_0 : $\pi_m > \hat{\Pi}(t_0) \geq \pi_h$

Se J_m acquisisce una risorsa a t_0 , nessun job con priorità maggiore o uguale può richiedere una risorsa già in uso a t_0

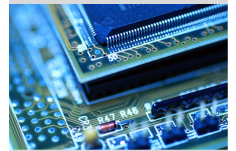


Unicità del job bloccante (2)

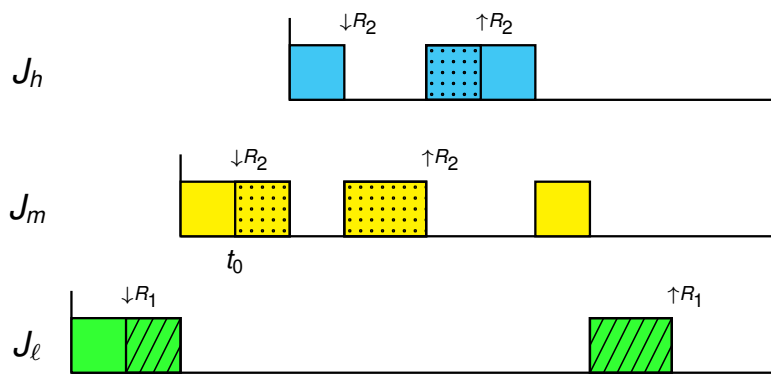


- $\pi_h > \pi_m > \pi_\ell \Rightarrow \Pi(R_1) \geq \pi_h$ e $\Pi(R_2) \geq \pi_m$
- $\hat{\Pi}(t_0) \geq \Pi(R_1) \geq \pi_h$
- Requisito per allocazione a t_0 : $\pi_m > \hat{\Pi}(t_0) \geq \pi_h$

J_h non può essere bloccato da J_ℓ se J_ℓ è stato interrotto da J_m e J_m ha acquisito una risorsa

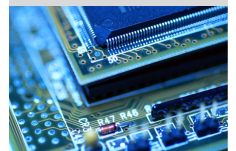


Unicità del job bloccante (3)

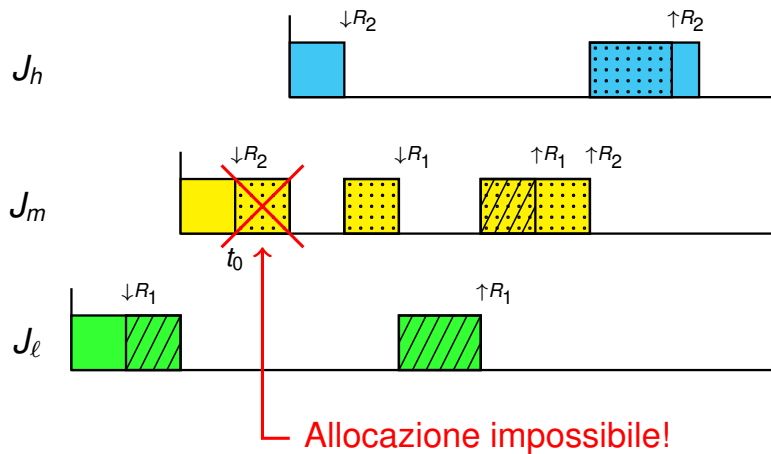


- $\pi_h > \pi_m > \pi_\ell \Rightarrow \Pi(R_1) \geq \pi_\ell$ e $\Pi(R_2) \geq \pi_h$
- $\hat{\Pi}(t_0) \geq \Pi(R_1) \geq \pi_\ell$
- Requisito per allocazione a t_0 : $\pi_m > \hat{\Pi}(t_0) \geq \pi_\ell$

J_h può essere bloccato da J_m solo se J_m possiede la risorsa che ha il massimo priority ceiling tra tutte quelle in uso ($= \hat{\Pi}(t)$)



Impossibilità del blocco transitivo



- $\pi_h > \pi_m > \pi_\ell \Rightarrow \Pi(R_1) \geq \pi_m \text{ e } \Pi(R_2) \geq \pi_h$
- $\hat{\Pi}(t_0) \geq \Pi(R_1) \geq \pi_m$
- Requisito per allocazione a t_0 : $\pi_m > \hat{\Pi}(t_0) \geq \pi_m$

Se J_m blocca J_h , J_m non può essere bloccato da J_ℓ

Tempo di blocco per conflitto di risorse

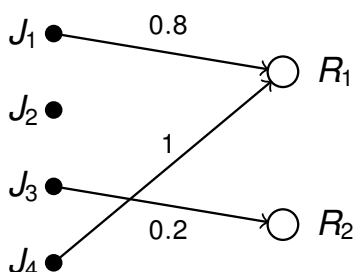
Il *tempo di blocco per conflitto di risorse* $b_i(rc)$ è il massimo ritardo di un job del task T_i causato da un conflitto di risorse

Come calcolare $b_i(rc)$ per il protocollo priority-ceiling?

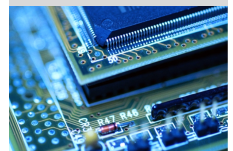
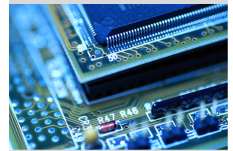
Con **priority ceiling** esistono 3 tipi di blocco: blocco diretto, blocco per **priority inheritance** e blocco per **priority ceiling**

Poiché ogni job è bloccato al massimo per la durata di una sola sezione critica, è sufficiente per ciascun task determinare i valori massimi dei ritardi introdotti da ciascun tipo di blocco

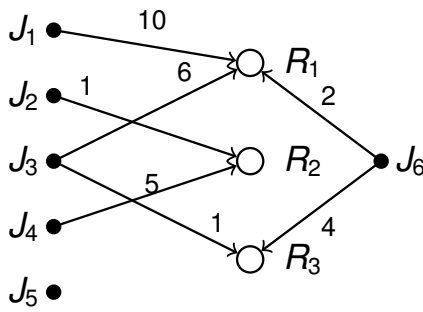
Esempio: $J_1: [R_1; 0.8]$, $J_2, J_3: [R_2; 0.2]$, $J_4: [R_1; 1]$



- J_4 può bloccare direttamente J_1 per 1 unità di tempo $\Rightarrow b_1(rc) = 1$
- J_4 può bloccare J_2 e J_3 quando acquisisce $R_1 \Rightarrow b_2(rc) = b_3(rc) = 1$
- Ovviamente $b_4(rc) = 0$



Tempo di blocco per conflitto di risorse (2)



Blocco diretto (B_d):

B_d	J_2	J_3	J_4	J_5	J_6
J_1		6			2
J_2	*		5		
J_3		*			4
J_4			*		
J_5				*	

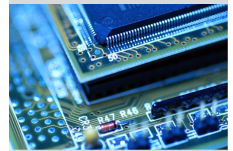
Blocco per inheritance (B_i):

B_i	J_2	J_3	J_4	J_5	J_6
J_1					
J_2	*	6			2
J_3		*	5		2
J_4			*		4
J_5				*	4

Blocco per ceiling (B_c):

B_c	J_2	J_3	J_4	J_5	J_6
J_1					
J_2	*	6			2
J_3		*	5		2
J_4			*		4
J_5				*	

- $B_i(r, c) = \max\{B_d(j, c) : 1 \leq j \leq r - 1\}$
- Se le priorità dei job sono tutte diverse, $B_c = B_i$ tranne che per i job che non utilizzano risorse (non bloccano)
- $b_i(rc) = \max_k\{B_d(i, k), B_i(i, k), B_c(i, k)\}$



Schedulabilità con priority-ceiling

Come si controlla la schedulabilità di un sistema che usa il protocollo priority-ceiling?

Applicando il test o le condizioni di schedulabilità ma considerando anche $b_i(rc)$ nel tempo di blocco del task T_i

Ad esempio, per la funzione di tempo richiesto si ha:

$$w_i(t) = e_i + b_i + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil e_k, \text{ ove}$$

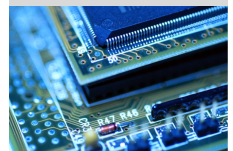
$$b_i = b_i(ss) + (K_i + 1) \cdot b_i(np) + (K_i + 1) \cdot b_i(rc), \text{ e}$$

K_i è il numero massimo di autosospensioni per un job di T_i

Come cambia l'overhead dovuto ai cambi di contesto?

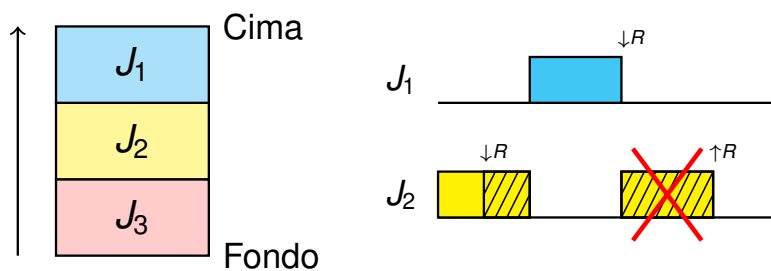
$$e'_i = e_i + 2 \cdot (K_i + 1) \cdot CS + 2 \cdot (K_i + 1) \cdot CS$$

(ultimo termine presente solo se il job usa risorse condivise!)

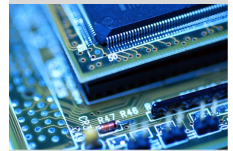


Protocollo stack-based priority-ceiling

- Protocollo **stack-based priority-ceiling** (Baker 1991)
- Semplificazione del protocollo priority-ceiling
- Versione base: ogni tipo di risorsa condivisa ha 1 unità
- Motivato da una esigenza particolare:
la condivisione di un unico stack da parte dei job
- Ogni job possiede una zona **contigua** dello stack
- Il job in esecuzione usa una zona in **cima** allo stack
- Lo spazio occupato da un job sullo stack è recuperato solo quando il job completa l'esecuzione



Nessun job deve bloccare o auto-suspendersi!



Protocollo stack-based priority-ceiling (2)

Per ogni risorsa R , $\Pi(R)$ definito come nel protocollo priority-ceiling

Regola di aggiornamento di $\hat{\Pi}(t)$

$\hat{\Pi}(t)$ è il massimo priority-ceiling tra tutte le risorse allocate, oppure Ω se tutte le risorse sono libere

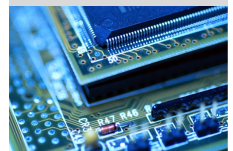
Regola di schedulazione

Non appena rilasciato, un job J con priorità assegnata π non può essere eseguito finché è vera la condizione $\pi \leq \hat{\Pi}(t)$

I job eseguibili sono schedulati in modo interrompibile in accordo alle priorità assegnate

Regola di allocazione

Quando un job richiede una risorsa, la richiesta è soddisfatta



Proprietà del protocollo stack-based priority-ceiling

Un job J_h può interrompere un job J_ℓ avente priorità più bassa?

Sì, però J_ℓ non può tornare in esecuzione prima che J_h termini

*Quando un job J comincia l'esecuzione, tutte le risorse che utilizza sono libere? **Si!***

Infatti inizia ad eseguire quando la sua priorità assegnata π diventa $\pi > \hat{\Pi}(t)$, ossia quando nessuna risorsa che utilizza è assegnata

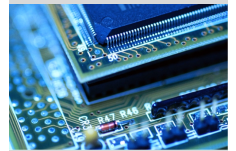
Nel protocollo **stack-based priority-ceiling** non si verificano mai deadlock

*Per la correttezza del protocollo è necessario che i job non si auto-suspendano? **Si!***

Il controllo d'accesso è effettuato solo al rilascio di un job e assume che il job non venga sospeso

Controllo d'accesso
alle risorse condivise

Marco Cesati



Schema della lezione

Priority-ceiling

Stack-based
priority-ceiling

Ceiling-priority

Auto-sospensione

Priorità dinamica

Job aperiodici

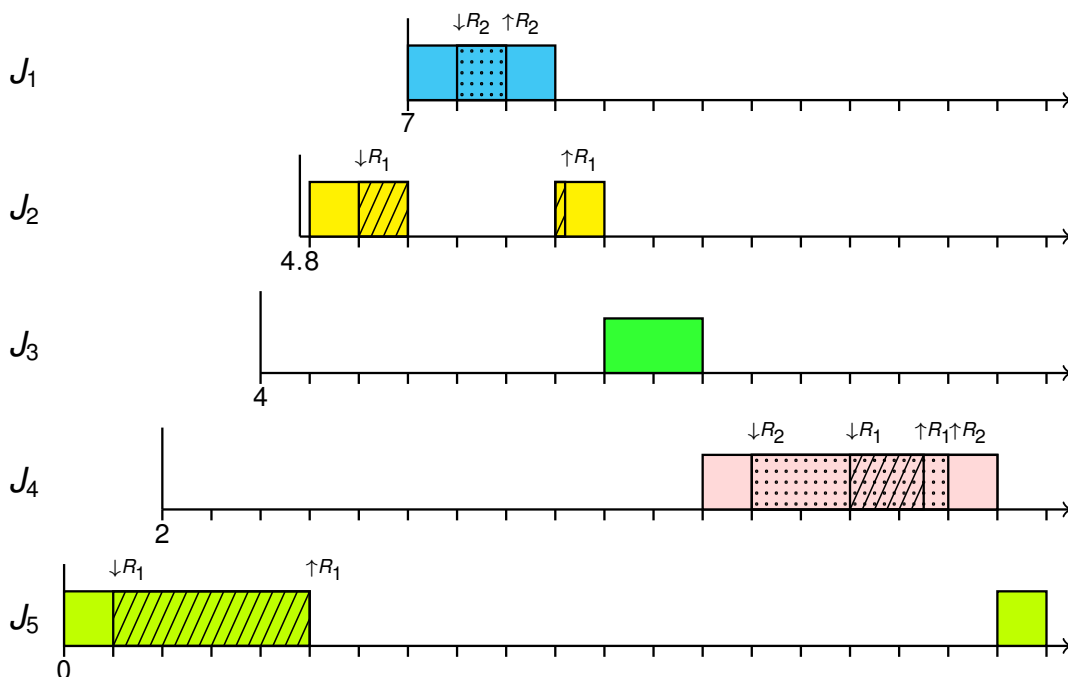
SERT'20

R10.15

Esempio di schedulazione con stack-based priority-ceiling

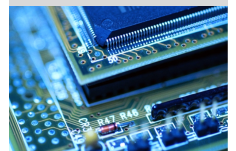
	J_1	J_2	J_3	J_4	J_5	$J_1: [R_2; 1]$	$J_2: [R_1; 1.2]$	$\Pi(R_1)=2$
r	7	4.8	4	2	0			
e	3	3	2	6	6	$J_4: [R_2; 4]$	$J_5: [R_1; 4]$	$\Pi(R_2)=1$

$\hat{\Pi}(t)$: Ω 2 Ω 2 1 2 Ω 1 Ω



Controllo d'accesso
alle risorse condivise

Marco Cesati



Schema della lezione

Priority-ceiling

Stack-based
priority-ceiling

Ceiling-priority

Auto-sospensione

Priorità dinamica

Job aperiodici

SERT'20

R10.16

Protocollo ceiling-priority

Utilizzato nel [Real-Time Systems Annex](#) di [Ada95](#)

Regole di schedulazione

- (a) Se un job non possiede alcuna risorsa, la sua priorità è quella assegnata dallo scheduler
 - (b) Se un job possiede una risorsa, la sua priorità è uguale al massimo [priority ceiling](#) di tutte le risorse assegnate al job
- Job con priorità identica sono schedulati in modo FIFO

Regola di allocazione

Quando un job richiede una risorsa, la richiesta è soddisfatta

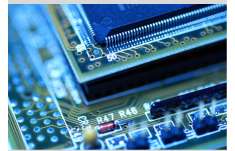
Ceiling-priority e stack-based priority-ceiling sono differenti?

Senza auto-sospensione producono schedulazioni identiche

Però è possibile modificare le regole di [ceiling-priority](#) per consentire l'auto-sospensione

Controllo d'accesso
alle risorse condivise

Marco Cesati



[Schema della lezione](#)

[Priority-ceiling](#)

[Stack-based
priority-ceiling](#)

[Ceiling-priority](#)

[Auto-sospensione](#)

[Priorità dinamica](#)

[Job aperiodici](#)

SERT'20

R10.17

Confronto tra priority-ceiling e stack-based priority-ceiling

- Nel caso peggiore i protocolli sono equivalenti:

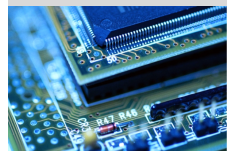
Teorema (Baker 1991)

I tempi di blocco massimi $b_i(rc)$ dovuti ai conflitti di risorse per [priority-ceiling](#) e per [stack-based priority-ceiling](#) sono identici

- Scheduler basati su [stack-based priority-ceiling](#) o [ceiling-priority](#) sono più semplici ed efficienti
- Scheduler basati su [stack-based priority-ceiling](#) o [ceiling-priority](#) hanno meno cambi di contesto
- I cambi di priorità dinamica sono meno frequenti in [priority-ceiling](#) perché si verificano solo in caso di effettiva contesa di una risorsa

Controllo d'accesso
alle risorse condivise

Marco Cesati



[Schema della lezione](#)

[Priority-ceiling](#)

[Stack-based
priority-ceiling](#)

[Ceiling-priority](#)

[Auto-sospensione](#)

[Priorità dinamica](#)

[Job aperiodici](#)

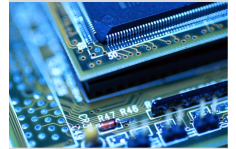
SERT'20

R10.18

Controllo d'accesso per job con auto-sospensione

Controllo d'accesso
alle risorse condivise

Marco Cesati



Schema della lezione

Priority-ceiling

Stack-based
priority-ceiling

Ceiling-priority

Auto-sospensione

Priorità dinamica

Job aperiodici

SERT'20

R10.19

I vari protocolli devono essere adattati in presenza di job che si auto-sospendono

- **NPCS**: non è possibile auto-sospendersi all'interno di una sezione critica
- **Priority-inheritance**: se un job J è bloccato su una risorsa posseduta da un job J' auto-sospeso, la priorità dinamica di J' è aggiornata solo se $\pi(t) > \pi'(t)$
- **Priority-ceiling**: stesse modifiche di **priority-inheritance**
- **Stack-based priority-ceiling**: l'auto-sospensione non è mai ammessa
- **Ceiling-priority**: se un job si auto-sospende in una sezione critica nessun job di priorità minore o uguale può essere eseguito

Esempio di priority-ceiling con auto-sospensione

Controllo d'accesso
alle risorse condivise

Marco Cesati



Schema della lezione

Priority-ceiling

Stack-based
priority-ceiling

Ceiling-priority

Auto-sospensione

Priorità dinamica

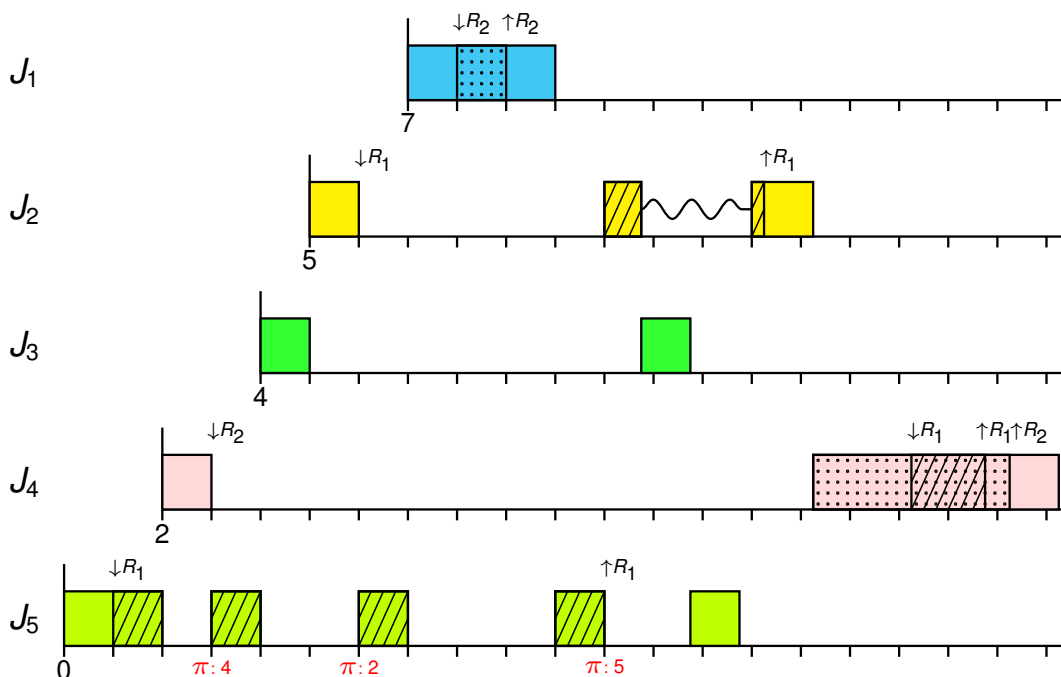
Job aperiodici

SERT'20

R10.20

	J_1	J_2	J_3	J_4	J_5	$J_1: [R_2; 1]$	$J_2: [R_1; 1]$	$\Pi(R_1)=2$
r	7	5	4	2	0			
e	3	3	2	6	6	$J_4: [R_2; 4]$	$[R_1; 1.5]$	$J_5: [R_1; 4]$
								$\Pi(R_2)=1$

$\hat{\pi}(t)$: Ω 2 1 2 Ω 1 Ω



Tempi di blocco con auto-sospensione

Considerando job con auto-sospensione:

- **NPCS:**

$$b_i = b_i(ss) + (K_i + 1) \cdot \max \{b_i(np), b_i(rc)\}$$

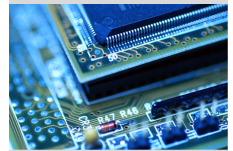
- **Priority-ceiling** e **ceiling-priority**:

$$b_i = b_i(ss) + (K_i + 1) \cdot (b_i(np) + b_i(rc))$$

Con priority-ceiling e ceiling-priority, i tempi di blocco $b_i(rc)$ devono essere calcolati considerando nella durata delle sezioni critiche anche gli eventuali tempi massimi di auto-sospensione (ossia, il numero di auto-sospensioni K_i moltiplicato per la durata massima dell'auto-sospensione x_i)

Controllo d'accesso
alle risorse condivise

Marco Cesati



Schema della lezione

Priority-ceiling

Stack-based
priority-ceiling

Ceiling-priority

Auto-sospensione

Priorità dinamica

Job aperiodici

SERT'20

R10.21

Priority-ceiling in sistemi a priorità dinamica

È possibile applicare i protocolli **priority-ceiling** e **ceiling-priority** anche a sistemi con priorità dinamica (Chen, Lin 1990)

Il valore **priority ceiling** di una risorsa non è più costante: dipende dalla priorità dinamica dei job che potenzialmente fanno uso della risorsa

Ad esempio, con **EDF** per ogni nuovo job rilasciato è necessario aggiornare:

- i valori numerici di priorità di tutti i job attivi
- i valori **priority ceiling** di tutte le risorse
- il valore del **current priority ceiling** di sistema

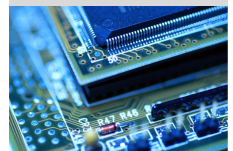
Ancora peggiore è il caso di una scheduler con priorità dinamica a livello di job!

Questi protocolli di controllo d'accesso funzionano, ma gli algoritmi sono complessi e l'implementazione costosa

Altri protocolli come NPCS o priority-inheritance sono più adatti a sistemi con priorità dinamica

Controllo d'accesso
alle risorse condivise

Marco Cesati



Schema della lezione

Priority-ceiling

Stack-based
priority-ceiling

Ceiling-priority

Auto-sospensione

Priorità dinamica

Job aperiodici

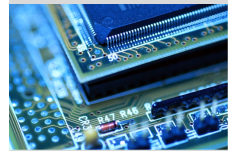
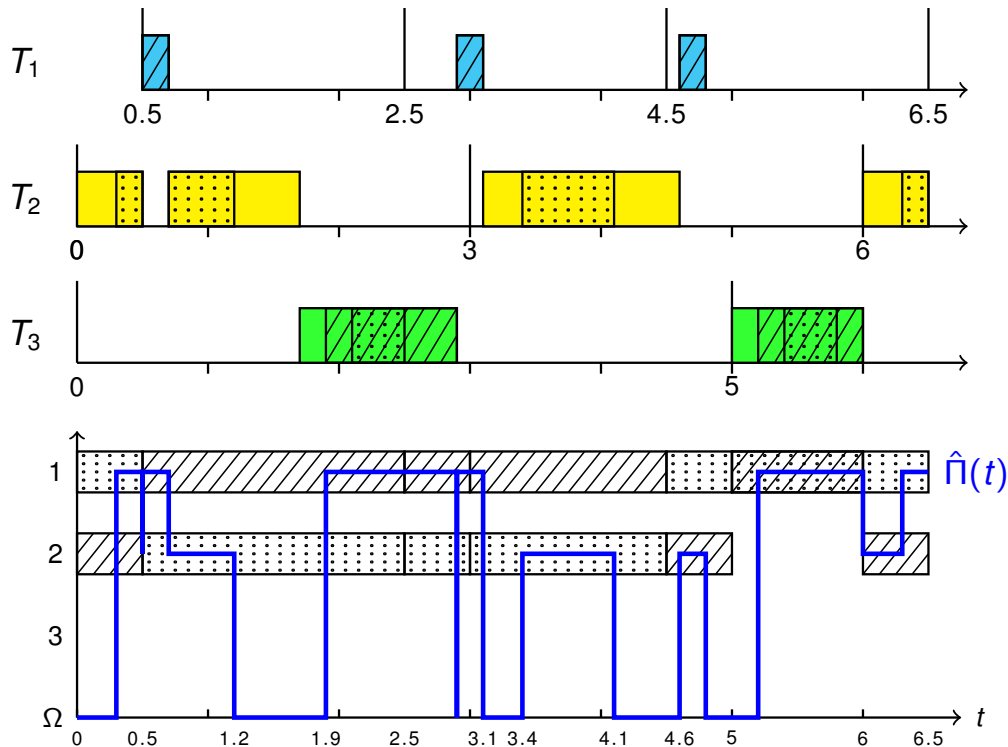
SERT'20

R10.22

Esempio di priority-ceiling con schedulazione EDF

$$T_1 = (0.5, 2, 0.2, 2; [R_1; 0.2]), \quad T_2 = (3, 1.5; [R_2; 0.7]),$$

$$T_3 = (5, 1.2; [R_1; 1[R_2; 0.4]])$$



Accesso alle risorse di job aperiodici

I protocolli di controllo d'accesso alle risorse sono utilizzabili anche con job aperiodici eseguiti in server periodici

Problema: un server procrastinabile esaurisce il budget mentre il job in esecuzione è entro una sezione critica

Soluzione:

- (1) L'esecuzione in una sezione critica del server periodico lo rende non interrompibile, anche se il budget è esaurito
- (2) L'eventuale ritardo del server periodico è recuperato assegnando corrispondentemente meno budget nei rifornimenti successivi
- (3) Nel controllare la schedulabilità si deve considerare tra i tempi di blocco anche il tempo d'esecuzione della più lunga sezione critica dei job aperiodici

