

Sistemi-Tempo-Reale

Pietro Focaccia

2024-2025

Contents

I	Hard Real Time	4
1	Periodici clock-driven	4
1.1	Test	4
1.1.1	Fattore di utilizzazione della CPU	4
1.2	Algoritmi	4
1.2.1	Timer-driven	4
1.2.2	Cycle executive	4
1.2.3	Baker Shaw	4
2	Periodici priority-driven	6
2.1	Test	6
2.1.1	Diagrammi temporali	6
2.1.2	Liu-Layland	6
2.1.3	Corollario Liu-Layland	6
2.1.4	Kuo-Mok	6
2.1.5	Burchard	6
2.1.6	Han	7
2.1.7	Audsley	7
2.1.8	Audsley v.2	8
2.2	Algoritmi	8
2.2.1	RMPO	8
3	Sporadici e periodici	8
3.1	Test	8
3.1.1	Diagrammi temporali	8
3.1.2	Audsley	8
3.1.3	Audsley v.2	9
3.1.4	Densità di utilizzazione del processore	9
3.1.5	Lehoczky	10
3.1.6	Lehoczky - utilizzazione efficace	10
3.2	Algoritmi	11
3.2.1	DMPO	11
4	Priorità dinamica	11
4.1	Test	11
4.1.1	Condizioni EDF	11
4.1.2	Test processor demand	12
4.2	Algoritmi	12
4.2.1	Earliest Deadline First	12
4.2.2	Least Slack Time First	12
II	Soft Real Time	12

5	Server a priorità statica	13
5.1	Polling Server	13
5.2	Deferrable Server	14
5.3	Priority Exchange Server	14
5.4	Sporadic Server	14
6	Server a priorità dinamica	15
6.1	Constant Utilization Server	15
6.2	Total Bandwidth Server	16
III	Risorse condivise	16
7	Protocolli	17
7.1	NOP (priorità statiche)	17
7.2	Non-Preemptive Critical Section Protocol	17
7.3	Priority Inheritance Protocol	17
7.4	Priority Ceiling Protocol	18
7.5	Immediate Priority Ceiling Protocol	19
7.6	Preemption Ceiling Protocol	19
7.7	Stack Resource Policy (una istanza)	19
7.8	Stack Resource Policy (+ istanze)	19
8	Test	21
8.1	Audsley (DMPO, RMPO)	21
8.2	Test EDF	21

Part I

Hard Real Time

1 Periodici clock-driven

1.1 Test

1.1.1 Fattore di utilizzazione della CPU

Condizione **sufficiente** affinché N processi siano schedulabili:

$$U = \sum_{i=1}^N U_i = \sum_{i=1}^N \frac{C_i}{T_i} \leq 1$$

1.2 Algoritmi

1.2.1 Timer-driven

A mano.

1.2.2 Cycle executive

Questo approccio è applicabile solamente ad insiemi di processi con periodi in relazione armonica tra loro, esempio:

- 25, 50, 100 in relazione armonica
- 20, 50, 100 non in relazione armonica

Si definisce il **ciclo minore** e **ciclo maggiore**, corrispondenti al periodo minore e maggiore. Poi si procede ad organizzare lo scheduling.

1.2.3 Baker Shaw

(può capitare che dica cycle executive per indicare questo)

Cerchiamo M (ciclo maggiore) e m (frame):

$$M = mcm(T_1, \dots, T_n)$$

invece m deve rispettare le condizioni:

- $M \bmod m = 0$
- $m \geq C_i \ \forall i$
- $m \leq T_i \ \forall i$
- $2m - M.C.D.(m, T_i) \leq T_i \ \forall i | (T_i \bmod m) > 0$

Una volta trovati M e m , stiliamo una tabella con i frame occupabili da ciascun job e poi eseguiamo la schedulazione.

Esempio

	C	T
P ₁	1	4
P ₂	2	5
P ₃	1	10
P ₄	2	20

$$M = 20, m = 2 \Rightarrow n_{cm} = 10 \quad n_{J1} = 5 \quad n_{J2} = 4 \quad n_{J3} = 2 \quad n_{J4} = 1$$

identificazione del ciclo minore (o dei cicli minori) in cui ciascun job può essere eseguito:

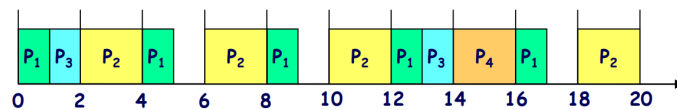
	J ₁₁	J ₁₂	J ₁₃	J ₁₄	J ₁₅	J ₂₁	J ₂₂	J ₂₃	J ₂₄	J ₃₁	J ₃₂	J ₄₁
c ₁	x					x				x		x
c ₂	x					x				x		x
c ₃		x								x		x
c ₄		x					x			x		x
c ₅			x				x			x		x
c ₆			x					x			x	x
c ₇				x				x			x	x
c ₈				x							x	x
c ₉					x				x		x	x
c ₁₀					x				x		x	x

0
2
4
6
8
10
12
14
16
18
20
↓ t

Nella realizzare la tabella successiva, in cui si fa lo scheduling, il numero di colonne coincide con m . Inserisci tutto quello che ci sta, poi dici i job che non ci stanno (es. $J_{21}, J_{22} \dots$).

criteri:

	1	1	1,3	1
c ₁	J ₁₁	J ₁₁	J ₁₁ J ₃₁	J ₁₁ J ₃₁
c ₂		J ₂₁	J ₂₁	J ₂₁
c ₃	J ₁₂	J ₁₂	J ₁₂	J ₁₂
c ₄		J ₂₂	J ₂₂	J ₂₂
c ₅	J ₁₃	J ₁₃	J ₁₃	J ₁₃
c ₆		J ₂₃	J ₂₃	J ₂₃
c ₇	J ₁₄	J ₁₄	J ₁₄ J ₃₂	J ₁₄ J ₃₂
c ₈				J ₄₁
c ₉	J ₁₅	J ₁₅	J ₁₅	J ₁₅
c ₁₀		J ₂₄	J ₂₄	J ₂₄



Quando chiede il rilassamento dei vincoli intende job slicing (es. $P_2(20, 4) \Rightarrow P'_2(20, 2), P''_2(20, 2)$ con $P'_2 \leftarrow P''_2$)

2 Periodici priority-driven

2.1 Test

2.1.1 Diagrammi temporali

Si cerca di schedulare i processi grazie ai diagrammi temporali e si testa se è possibile.

2.1.2 Liu-Layland

Condizione sufficiente alla schedulabilità di N processi con RMPO:

$$U \leq N(2^{1/N} - 1)$$

2.1.3 Corollario Liu-Layland

Test meno stringente del precedente:

$$U_{RMPO} = \prod_{i=1}^N (1 + U_i) \leq 2$$

2.1.4 Kuo-Mok

Si raggruppano i task con i periodi in relazione armonica tra loro, il nuovo task creato avrà:


- $U_{nuovo} = U_x + U_y + \dots + U_z$
- $T_{nuovo} = \min(T_x, T_y, \dots, T_z)$
- $C_{nuovo} = U_{nuovo} * T_{nuovo}$

I nuovi task di sottopongono poi al test di Liu-Layland o al suo corollario.

N.B. se il partizionamento non è univoco va preferito quello con fattori di utilizzazione più disuniformi.

Esempio

S	C	T	C/T
P ₁	4	10	0.40
P ₂	4	20	0.20
P ₃	8	40	0.20
P ₄	3.6	45	0.08
P ₅	1.8	90	0.02



$P_1' \equiv \{P_1, P_2, P_3\}$
 $P_2' \equiv \{P_4, P_5\}$

S'	C'	T'	C'/T'
P ₁ '	8	10	0.8
P ₂ '	4.5	45	0.1

2.1.5 Burchard

Per prima cosa dobbiamo calcolare: $X_j = \log_2 T_j - \lfloor \log_2 T_j \rfloor \forall j$.

Successivamente dobbiamo calcolare la distorsione: $\zeta = \max x_i - \min x_j$ con $1 \leq j \leq N$ e $1 \leq i \leq N$.

Ottenuto questo valore il coefficiente di utilizzazione massimo deve essere:

$$U_{\text{RMPO}}(N, \zeta) = \begin{cases} (N-1)(2^{\zeta/(N-1)} - 1) + 2^{1-\zeta} - 1 & \zeta < 1 - 1/N \\ N(2^{1/N} - 1) & \zeta \geq 1 - 1/N \end{cases}$$

2.1.6 Han

Fornisce una condizione **sufficiente**. Si crea un **insieme accelerato** di processi, ovvero un insieme dove i tempi di calcolo non variano ma i periodi sono tutti \leq . Se l'**insieme accelerato** è schedulabile allora lo è anche quello normale.

Per creare l'insieme accelerato si parte dal task con il periodo minore e si cerca di mettere gli altri in relazione armonica, per poi applicare i test (es. somma U). Se con il primo fallisce si prende il secondo e così via.

Se non riesce subito provare qualche combinazione.

2.1.7 Audsley

Condizione **sufficiente**:

$$R_i = C_i + I_i(R_i) \leq T_i \quad i = 1, \dots, N$$

Dove I_i è l'interferenza sul tempo di risposta R_i del processo i -esimo, dovuta ai processi a priorità maggiore.

$$I_i(R_i) = \sum_{j|p_j > p_i} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Esempio

A_5	C	T	C/T
P_1	5	10	0.50
P_2	8	19	0.42

$$U(A_5) = 0.92$$

$$\text{processo } P_1: R_1^1 = R_1^0 = C_1 = 5 < T_1 = 10$$

$$\text{processo } P_2: R_2^0 = C_2 = 8$$

$$R_2^1 = C_2 + \lceil R_2^0/T_1 \rceil * C_1 = 8 + \lceil 8/10 \rceil * 5 = 13$$

$$R_2^2 = C_2 + \lceil R_2^1/T_1 \rceil * C_1 = 8 + \lceil 13/10 \rceil * 5 = 18$$

$$R_2^3 = C_2 + \lceil R_2^2/T_1 \rceil * C_1 = 8 + \lceil 18/10 \rceil * 5 = 18$$

$$R_2^3 = R_2^2 = 18 < T_2 = 19$$

2.1.8 Audsley v.2

Condizione **sufficiente**, meno oneroso di calcoli, però fallisce più spesso:

$$C_i + I_i(T_i) = C_i + \sum_{j|p_j > p_i} \left\lceil \frac{R_j}{T_j} \right\rceil C_j \leq T_i$$

Si applica questo a tutti i processi, per quelli che falliscono si applica **Audsley**

Esempio

A₅	C	T	C/T	U(A₅) = 0.92
P₁	5	10	0.50	
P₂	8	19	0.42	

processo P₁: C₁ + I₁(T₁) = 5 < T₁ = 10

processo P₂: C₂ + ⌈T₂/T₁⌉ * C₁ = 8 + ⌈19/10⌉ * 5 = 18 < T₂ = 19

2.2 Algoritmi

2.2.1 RMPO

Ad ogni processo associata una **priorità statica, direttamente proporzionale** alla sua frequenza di esecuzione.

Algoritmo ottimo, se un insieme di job non è schedulabile con lui non lo è con nessuno.

3 Sporadici e periodici

3.1 Test

3.1.1 Diagrammi temporali

Si cerca di schedulare i processi grazie ai diagrammi temporali e si testa se è possibile.

3.1.2 Audsley

Condizione **necessaria** e **sufficiente** affinché N task siano schedulabili con **DMPO**:

$$R_i = C_i + I_i(R_i) = C_i + \sum_{j|p_j > p_i} \left\lceil \frac{R_j}{T_j} \right\rceil C_j \leq D_j \quad i = 1, \dots, N$$

Esempio

A_6	C	T	D	C/T
P ₁	4	10	10	0.40
P ₂	3	15	6	0.20
P ₃	6	22	22	0.27

$$U(A_6) = 0.87$$

$$P_2: R_2^1 = R_2^0 = C_2 = 3 < D_2 = 6$$

$$P_1: R_1^0 = C_1 = 4$$

$$R_1^1 = C_1 + \lceil R_1^0/T_2 \rceil * C_2 = 4 + \lceil 4/15 \rceil * 3 = 7$$

$$R_1^2 = C_1 + \lceil R_1^1/T_2 \rceil * C_2 = 4 + \lceil 7/15 \rceil * 3 = 7$$

$$R_1^2 = R_1^1 = 7 < D_1 = 10$$

$$P_3: R_3^0 = C_3 = 6$$

$$R_3^1 = C_3 + \lceil R_3^0/T_2 \rceil * C_2 + \lceil R_3^0/T_1 \rceil * C_1 = 6 + \lceil 6/15 \rceil * 3 + \lceil 6/10 \rceil * 4 = 13$$

$$R_3^2 = C_3 + \lceil R_3^1/T_2 \rceil * C_2 + \lceil R_3^1/T_1 \rceil * C_1 = 6 + \lceil 13/15 \rceil * 3 + \lceil 13/10 \rceil * 4 = 17$$

$$R_3^3 = C_3 + \lceil R_3^2/T_2 \rceil * C_2 + \lceil R_3^2/T_1 \rceil * C_1 = 6 + \lceil 17/15 \rceil * 3 + \lceil 17/10 \rceil * 4 = 20$$

$$R_3^4 = C_3 + \lceil R_3^3/T_2 \rceil * C_2 + \lceil R_3^3/T_1 \rceil * C_1 = 6 + \lceil 20/15 \rceil * 3 + \lceil 20/10 \rceil * 4 = 20$$

$$R_3^4 = R_3^3 = 20 < D_3 = 22$$

3.1.3 Audsley v.2

Alternativa più rapida, ma solo **sufficiente**:

$$C_i + I_i(D_i) = C_i + \sum_{j|p_j > p_i} \left\lceil \frac{D_i}{T_j} \right\rceil C_j \leq D_j \quad i = 1, \dots, N$$

Applico questa formula per tutti i task e la precedente per quelli che falliscono.

Esempio

A_6	C	T	D	C/T
P ₁	4	10	10	0.40
P ₂	3	15	6	0.20
P ₃	6	22	22	0.27

$$U(A_6) = 0.87$$

$$P_2: C_2 + I_2(D_2) = 3 < D_2 = 6$$



$$P_1: C_1 + \lceil D_1/T_2 \rceil * C_2 = 4 + \lceil 10/15 \rceil * 3 = 7 < D_1 = 10$$



$$P_3: C_3 + \lceil D_3/T_2 \rceil * C_2 + \lceil D_3/T_1 \rceil * C_1 = 6 + \lceil 22/15 \rceil * 3 + \lceil 22/10 \rceil * 4 = 24 > D_3 = 22$$



3.1.4 Densità di utilizzazione del processore

Condizione **sufficiente**, basata sul calcolo della densità di utilizzazione del processore:

$$\Delta = \sum_{j=1}^N \frac{C_j}{D_j} \leq U_{RMPO}(N) = N(2^{\frac{1}{N}} - 1)$$

3.1.5 Lehoczky

Condizione **sufficiente**, definiamo $D_j = \delta_j * T_j$, allora:

$$\sum_{j=1}^N \frac{C_j}{T_j} \leq U(N, \delta) = \begin{cases} N((2\delta)^{1/N} - 1) + 1 - \delta & 0.5 \leq \delta \leq 1 \\ \delta & 0 \leq \delta \leq 0.5 \end{cases} \quad \delta = \min_j \{\delta_j\}$$

Esempio

	C	T	D	C/T	D/T
P ₁	1	4	3	0.25	0.75
P ₂	1	5	5	0.20	1
P ₃	3	15	10.5	0.20	0.7

$$\sum_{j=1}^3 \frac{C_j}{T_j} = 0.65 \leq U(N=3, \delta=0.7) = 0.656$$

3.1.6 Lehoczky - utilizzazione efficace

Calcoliamo il fattore di utilizzo efficace:

$$f_j = \left(\sum_{i \in H_n} \frac{C_i}{T_i} \right) + \frac{1}{T_j} \left(C_j + \sum_{k \in H_1} C_k \right)$$

- H_1 , insieme dei processi che possono interferire al più una volta;
- H_n , insieme dei processi che possono interferire due o più volte;

$$f_j \leq U(N = |H_n| + 1, \delta = \delta_j)$$

condizione che deve essere verificata $\forall P_j$.

Se $|H_1| = 0$ allora mettiamo comunque U del processo che stiamo analizzando.

Esempio

	C	T	D	C/T	C/D	D/T
P ₁	1	10	10	0.100	0.100	1
P ₂	4	12	12	0.333	0.333	1
P ₃	4	15	6	0.267	0.667	0.400
P ₄	1	30	15	0.033	0.067	0.500
P ₅	5	60	29	0.083	0.172	0.483

$$\Delta = 1.339 > U_{RMPO}(N=5) = 0.743$$

$$U = 0.817 > U(N=5, \delta=0.4) = 0.4$$

	H _n	H ₁	f	U(N,δ)	
P ₃	{ }	{ }	4/15=0.267	U(1,0.4)=0.4	👍
P ₁	{ }	{P ₃ }	(1+4)/10=0.5	U(1,1)=1	👍
P ₂	{P ₁ }	{P ₃ }	1/10+(4+4)/12=0.767	U(2,1)=0.828	👍
P ₄	{P ₁ ,P ₂ }	{P ₃ }	1/10+4/12+(1+4)/30=0.6	U(3,0.5)=0.5	👎
P ₅	{P ₃ ,P ₁ ,P ₂ }	{P ₄ }	4/15+1/10+4/12+(5+1)/60=0.8	U(4,0.483)=0.483	👎

3.2 Algoritmi

3.2.1 DMPO

Ad ogni processo associata una **priorità statica**, **inversamente proporzionale** alla sua deadline relativa.

Algoritmo ottimo, se un insieme di job non è schedulabile con lui non lo è con nessuno.

4 Priorità dinamica

4.1 Test

4.1.1 Condizioni EDF

Condizione **necessaria** e **sufficiente** affinché N task **periodici** siano schedulabili con EDF:

$$U = \sum_{i=1}^N \frac{C_i}{T_i} \leq 1.$$

Condizione **sufficiente** affinché N task **periodici** e **sporadici** siano schedulabili con EDF:

$$\Delta = \sum_{i=1}^N \frac{C_i}{D_i} \leq 1.$$

4.1.2 Test processor demand

A_7	C	T	D	C/T	C/D
P_1	4	10	10	0.40	0.40
P_2	3	15	6	0.20	0.50
P_3	7	22	22	0.32	0.32

$U(A_7) = 0.92$

$\Delta(A_7) = 1.22$

$$BI^0 = C_1 + C_2 + C_3 = 14$$

$$BI^1 = \lceil BI^0/T_1 \rceil C_1 + \lceil BI^0/T_2 \rceil C_2 + \lceil BI^0/T_3 \rceil C_3 = \lceil 14/10 \rceil 4 + \lceil 14/15 \rceil 3 + \lceil 14/22 \rceil 7 = 18$$

$$BI^2 = \lceil BI^1/T_1 \rceil C_1 + \lceil BI^1/T_2 \rceil C_2 + \lceil BI^1/T_3 \rceil C_3 = \lceil 18/10 \rceil 4 + \lceil 18/15 \rceil 3 + \lceil 18/22 \rceil 7 = 21$$

$$BI^3 = \lceil BI^2/T_1 \rceil C_1 + \lceil BI^2/T_2 \rceil C_2 + \lceil BI^2/T_3 \rceil C_3 = \lceil 21/10 \rceil 4 + \lceil 21/15 \rceil 3 + \lceil 21/22 \rceil 7 = 25$$

$$BI^4 = \lceil BI^3/T_1 \rceil C_1 + \lceil BI^3/T_2 \rceil C_2 + \lceil BI^3/T_3 \rceil C_3 = \lceil 25/10 \rceil 4 + \lceil 25/15 \rceil 3 + \lceil 25/22 \rceil 7 = 32$$

$$BI^5 = \lceil BI^4/T_1 \rceil C_1 + \lceil BI^4/T_2 \rceil C_2 + \lceil BI^4/T_3 \rceil C_3 = \lceil 32/10 \rceil 4 + \lceil 32/15 \rceil 3 + \lceil 32/22 \rceil 7 = 39$$

$$BI^6 = \lceil BI^5/T_1 \rceil C_1 + \lceil BI^5/T_2 \rceil C_2 + \lceil BI^5/T_3 \rceil C_3 = \lceil 39/10 \rceil 4 + \lceil 39/15 \rceil 3 + \lceil 39/22 \rceil 7 = 39 = BI^5$$

$$BI = 39 < H = \text{mcm}(10, 15, 22) = 330$$

$$\mathcal{D} \equiv \{6, 10, 20, 21, 22, 30, 36\}$$

t	$C_1(0, t)$	$C_2(0, t)$	$C_3(0, t)$	$C_p(0, t)$	$\leq t$
6	0	3	0	3	👍
10	4	3	0	7	👍
20	8	3	0	11	👍
21	8	6	0	14	👍
22	8	6	7	21	👍
30	12	6	7	25	👍
36	12	9	7	28	👍



4.2 Algoritmi

4.2.1 Earliest Deadline First

Ad ogni processo viene associata una **priorità maggiore**, quanto più è imminente la sua **deadline assoluta**

4.2.2 Least Slack Time First

Priorità maggiore ai task con **slack time inferiore**. (strict = slack time calcolato ad ogni clock) (non strict = calcolato ad ogni rilascio)

Part II

Soft Real Time

Ai task **HRT** si aggiungono quelli **SRT**, che non hanno delle deadline da rispettare, il nostro obbiettivo comunque è velocizzare la loro esecuzione.

Soluzione più semplice è **background**, ovvero eseguire i task **SRT** quando non sono in esecuzione quelli **HRT**. I tempi di risposta non sono ottimi, però non vengono alterati gli **HRT**. Per i diagrammi del background, se non ci sono **SRT** allora non scrivi niente, se ci sono e tendono di eseguire waiting.

Alternativa, generare un nuovo processo **HRT**(Server) che si occupa della gestione degli **SRT**, da dimensionare correttamente per non compromettere la schedulabilità.

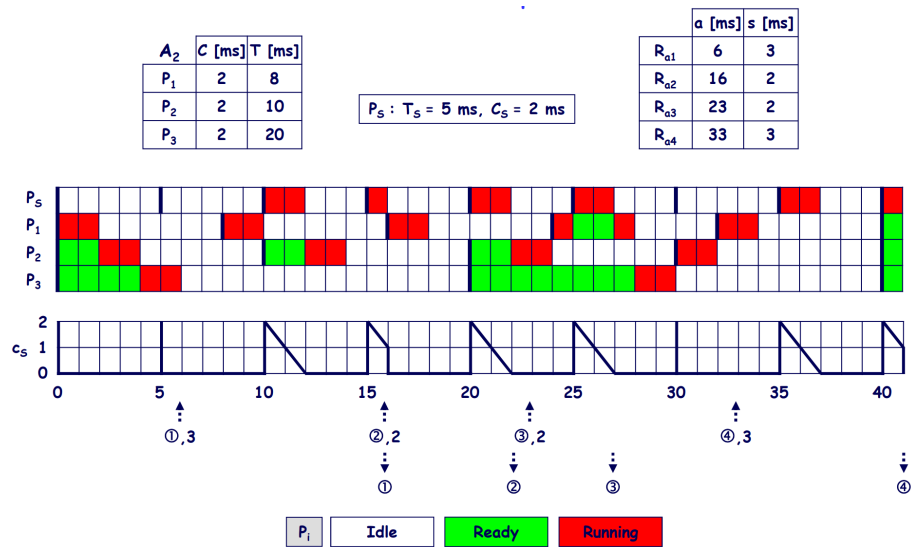
N.B.: può chiedere tabella con f_i (istante di completamento del task SRT i-esimo) e $f_i - a_i - c_i$ (lag di risposta).

5 Server a priorità statica

I task periodici, tra cui il server, sono schedulati tramite priorità statica (RMPO). Il server possiede i parametri T_s e C_s , da dimensionare correttamente per non interferire con gli **HRT**.

5.1 Polling Server

Ad ogni periodo T_s viene assegnato un tempo di computazione C_s , consumato dall'esecuzione dei task aperiodici ed inizializzato ad ogni ciclo, se non sono presenti task **SRT** il tempo viene buttato.



N.B.:

- La capacità va a zero quando non ci sono task SRT e quando il server prende il controllo (ATTENZIONE quando il server non è a massima priorità);
- Se il server non è a massima priorità però ci sono task SRT allora la capacità si conserva;

Per dimensionare il server:

$$U_S \leq (N+1)(2^{\frac{1}{N+1}} - 1) - U_p$$

Alternativa meno conservativa, che prende il server come task a massima priorità:

$$U_s \leq \frac{2}{\left(1 + \frac{U_p}{N}\right)^N} - 1$$

5.2 Deferrable Server

Come il Polling Server, però in caso di assenza di task aperiodici C_s viene conservato.

Per dimensionare il server:

$$U_s \leq \frac{2 - \left(1 + \frac{U_p}{N}\right)^N}{2 \left(1 + \frac{U_p}{N}\right)^N - 1}$$

5.3 Priority Exchange Server

- C_s può essere accumulata ad ogni livello di priorità inferiore a quella del server;
- ogni T_s viene ripristinata la capacità C_s al livello di priorità del server, le capacità agli altri livelli vengono mantenute;
- se non sono presenti **SRT** ed è in esecuzione un task **HRT** di priorità inferiore C_s viene trasferita al suo livello;
- se presente un **SRT** e il server possiede C_s allora avrà priorità ad ogni livello in cui ha C_s ;
- se hai tempo di calcolo su più livelli usi prima quello a maggior priorità;
- il server è in waiting quando non ha task, ma ha capacità. In ready quando ha task srt, ma non ha massima priorità;

Per dimensionare il server:

$$U_S \leq (N+1)(2^{\frac{1}{N+1}} - 1) - U_p$$

Alternativa meno conservativa, che prende il server come task a massima priorità:

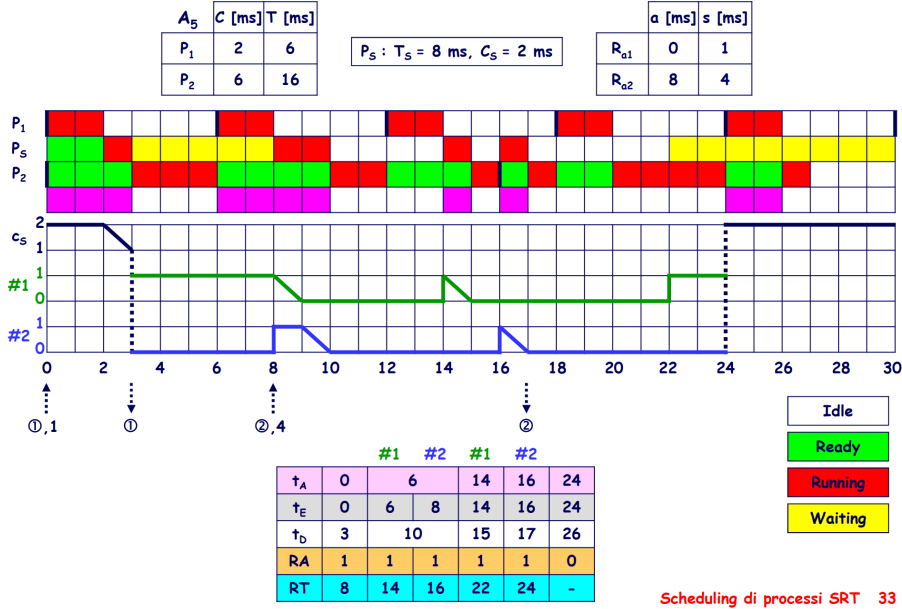
$$U_s \leq \frac{2}{\left(1 + \frac{U_p}{N}\right)^N} - 1$$

5.4 Sporadic Server

Definiamo:

- $t_a :=$ istante in cui $C_s > 0$ e P_s attivo (in esecuzione processo con priorità $>$ di P_s o server);
- $t_d :=$ istante in cui $C_s = 0$ o P_s non attivo;
- $t_e := \max\{RT, t_a\}$ $t_e := \max\{\max\{t_a + T_s, t_d\}, t_a\}$

- $RA :=$ capacità consumata in $[t_e, t_d]$
- $RT := \max\{t_e + T_s, t_d\}$



Per dimensionare il server:

$$U_S \leq (N+1)(2^{\frac{1}{N+1}} - 1) - U_p$$

Alternativa meno conservativa, che prende il server come task a massima priorità:

$$U_s \leq \frac{2}{\left(1 + \frac{U_p}{N}\right)^N} - 1$$

6 Server a priorità dinamica

I task periodici, tra cui il server, sono schedulati tramite priorità dinamica (EDF). Il server possiede i parametri D_s e C_s , da dimensionare correttamente per non interferire con gli **HRT**.

6.1 Constant Utilization Server

Ogni volta che arriva un **SRT** vengono calcolati C_s e D_s :

$$C_s = C_{RA}$$

$$D_s = t_{RA} + \frac{C_s}{U_s}$$

con t_{RA} tempo di arrivo del task aperiodico.

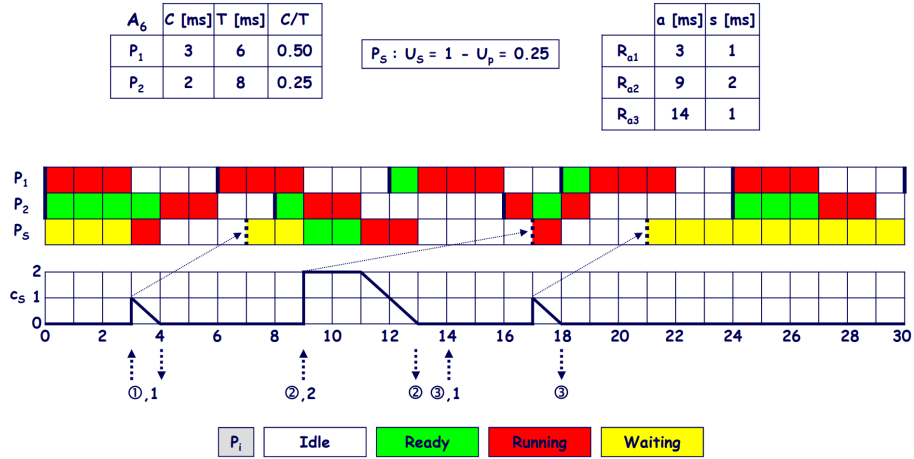
N.B.:

- se D_s viene con la virgola tieni un valore superiore;

- se una d supera a successivo, per il calcolo di d successivo usare d precedente non a del task, in questo caso tenere valore di d con la virgola;
- scrivi i calcoletti nella tabella;

Poi il server viene eseguito in accorto con la priorità data in base ai suoi parametri da EDF.

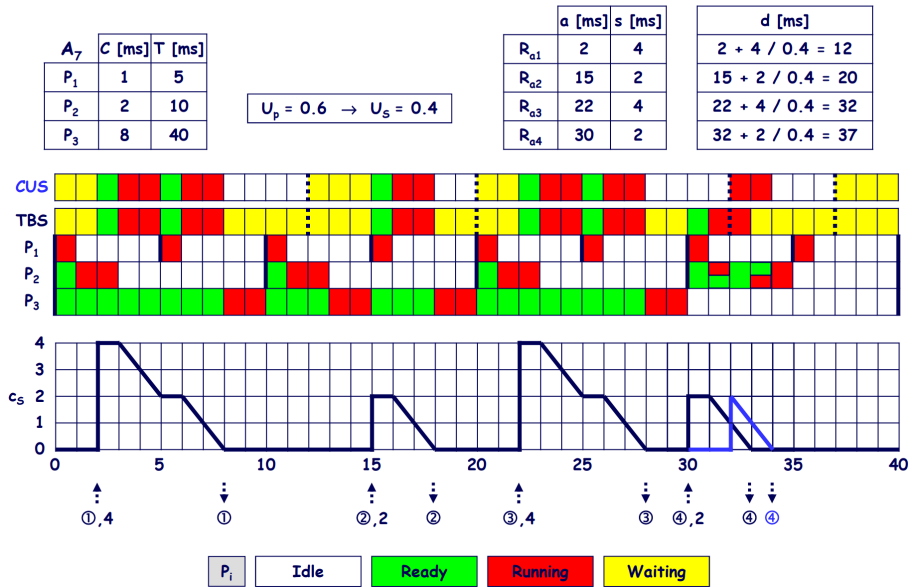
Anche se un task viene completato in anticipo un nuovo **SRT** non viene preso in considerazione fino a che non passa la deadline del precedente.



6.2 Total Bandwith Server

Uguale a CUS, a differenza che i nuovi **SRT** vengono presi in carico anche prima della deadline precedente.

Server in waiting se non ha task **SRT**.



Part III

Risorse condivise

7 Protocolli

7.1 NOP (priorità statiche)

- i task accedono alle risorse in base alle loro priorità;
- se ci sono risorse annidate quella esterna va considerata occupata quando si usa quella interna;

7.2 Non-Preemptive Critical Section Protocol

Con questo protocollo nessun processo all'interno della sezione critica può subire preemption, quindi quando entra nella sezione critica la sua priorità corrente diventa:

$$\pi_i = p_0 > \max\{p_1, p_2, \dots, p_N\}$$

quando termina la sezione critica torna alla sua priorità nominale.

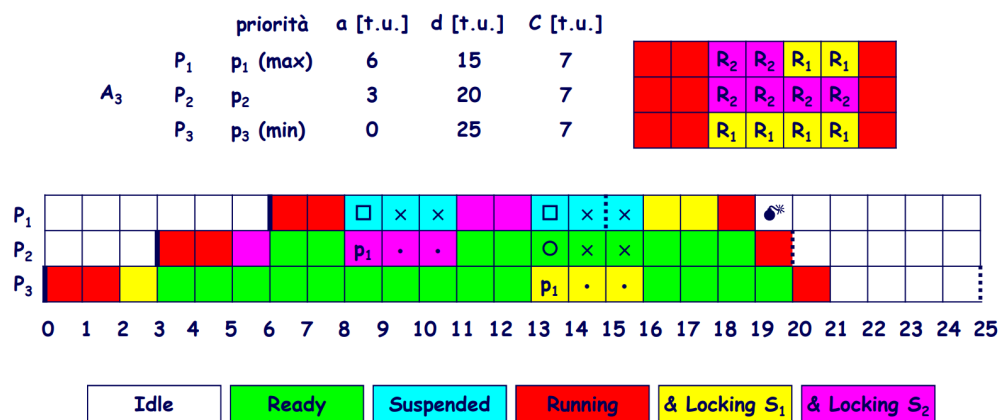
- x (blocked) se task ha max priorità, ma non può accedere alle risorse;
- - (suspended) se non ha max priorità;

Caratteristiche protocollo:

- non serve conoscere a priori le risorse usate dai vari processi
- previene inversioni di priorità incontrollate
- previene la concatenazione di blocchi
- previene situazioni di deadlock

7.3 Priority Inheritance Protocol

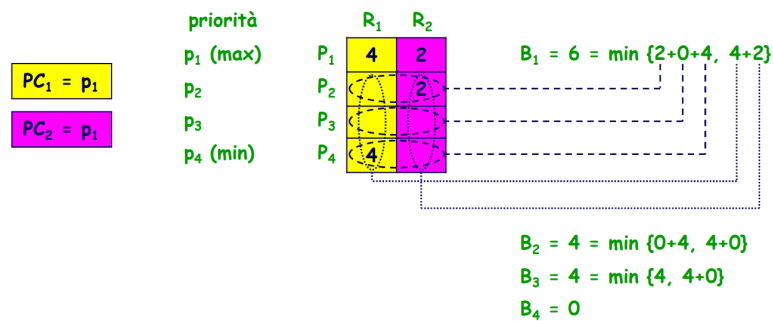
In questo protocollo un processo mantiene la sua priorità nominale se non detiene risorse, se le detiene eredita la priorità del processo a maggior priorità bloccato.



Caratteristiche protocollo (solo quelle diverse del precedente):

- non previene la concatenazione di blocchi
- non previene situazioni di deadlock

Calcolo dei blocchi:

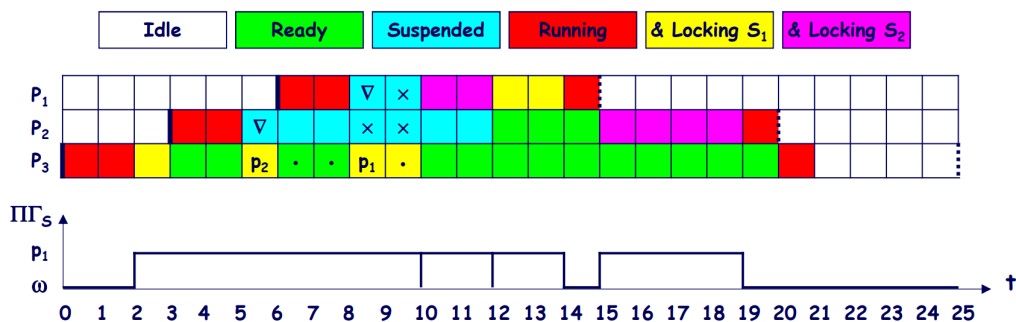


Nel completare la tabella con i vari blocchi, se ci sono accessi annidati nella risorsa esterna vanno considerate anche quelle interne.

7.4 Priority Ceiling Protocol

In questo protocollo un processo mantiene la sua priorità nominale se non detiene risorse, se le detiene eredita la priorità del processo a maggior priorità che può bloccare. A differenza del precedente l'accesso ad una risorsa viene negato se la priorità non è superiore a quella del tetto del sistema, anche se la risorsa è libera.

		priorità	a [t.u.]	d [t.u.]	C [t.u.]		
A_3	P_1	p_1 (max)	6	15	7		$PC_1 = p_1$
	P_2	p_2	3	20	7		$PC_2 = p_1$
	P_3	p_3 (min)	0	25	7		



Caratteristiche protocollo:

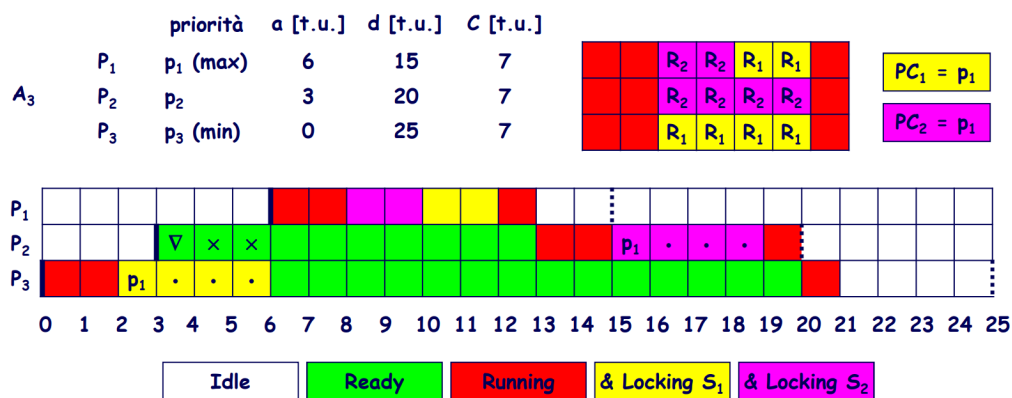
- serve conoscere a priori le risorse usate dai vari processi
- previene inversioni di priorità incontrollate

- previene la concatenazione di blocchi
- previene situazioni di deadlock

Per calcolare il blocco prendo il numero di blocco più grande, con priorità inferiore.

7.5 Immediate Priority Ceiling Protocol

Come il precedentem però ad un processo che detiene una risorsa viene immediatamente associata la priorità del ceiling, senza aspettare eventuali conflitti.



Caratteristiche protocollo (come le precedenti).

7.6 Preemption Ceiling Protocol

7.7 Stack Resource Policy (una istanza)

- scheduling con edf, priorità alla deadline più vicina;
- per ogni risorsa si stabilisce il ceiling, priorità maggiore tra i task che ci devono accedere;
- un task va in esecuzione solo se la sua priorità è maggiore del ceiling;

7.8 Stack Resource Policy (+ istanze)

Come prima, si aggiunge:

- u_k # di unità disponibili;
- n_{jk} # max di unità richieste da ogni task;
- v_k # di unità libere in un determinato istante;

Tetto di preemption di R_k dipendente da $v_k = \max$ livello dei task che chiedono più di v_k .

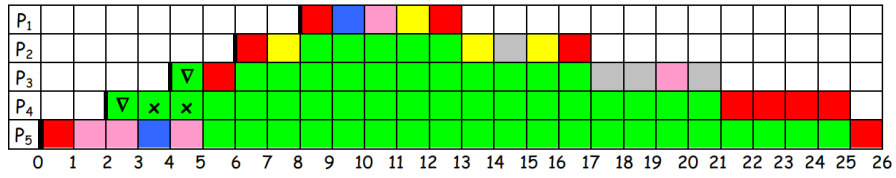
Per calcolare il blocco prendo il numero di blocco più grande, con priorità inferiore.

	C_{type}					
P_1			R_3	R_2	R_1	
P_2		R_1	R_1	R_4	R_1	
P_3		R_4	R_4	R_2	R_4	
P_4						
P_5		R_2	R_2	R_3	R_2	

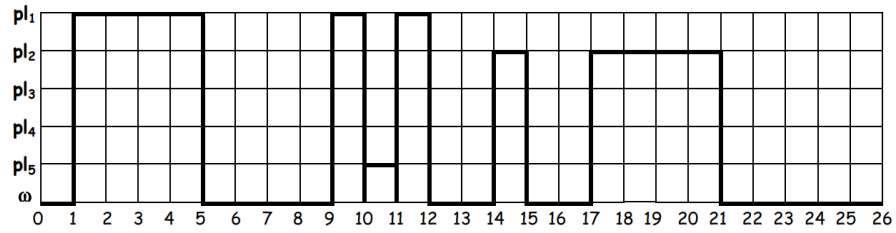
	R_1	R_2	R_3	R_4
P_1	2	1	1	
P_2	1			1
P_3		2		1
P_4				
P_5		3	1	

P_1	$pl_1(\max)$
P_2	pl_2
P_3	pl_3
P_4	pl_4
P_5	$pl_5(\min)$

	$v_k=0$	$v_k=1$	$v_k=2$	$v_k=3$
R_1	pl_1	pl_1	ω	ω
R_2	pl_1	pl_3	pl_5	ω
R_3	pl_1	ω	-	-
R_4	pl_2	ω	-	-



$\Pi\Lambda_s$



	R_1	R_2	R_3	R_4	B_i
P_1	1	1	1		4
P_2	4			1	4
P_3		1		4	4
P_4					4
P_5		4	1		0

8 Test

8.1 Audsley (DMPO, RMPO)

Audsley normale, con aggiunta del tempo di blocco massimo.

$$R_i = C_i + B_i + I_i(R_i) \leq D_i \quad \forall i$$

$$R_i^0 = C_i + B_i \quad \forall i, \quad R_i^n = C_i + B_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i^{n-1}}{T_k} \right\rceil C_k \quad i \neq 1, \quad n = 1, 2, \dots$$

8.2 Test EDF

$$\forall i, 1 \leq i \leq N, \sum_{k=1}^i \frac{C_k}{D_k} + \frac{B_i}{D_i} \leq 1$$