Hard Real Time Task Scheduling

back

Link Piter

Indice

Assunti

N processi P_i con $i = 1, 2, ..., N$ indipendenti

- Senza vincoli di precedenza
- Senza risorse condivise

Ogni processo P_j con $j = 1, 2, ..., N$

- è periodico, con periodo T_j prefissato
- è caratterizzato da un tempo massimo di esecuzione C_j con $C_j < T_j$
- è caratterizzato da una deadline D_j con $D_j = T_j$

L'esecuzione dei processi è affidata a un sistema di elaborazione monoprocessore. Il tempo impiegato dal processore per operare una commutazione di contesto tra processi è trascurabile.

Teorema sulla schedulabilità

> Condizione necessaria perchè N precessi siano schedulabili
>
> $U = \sum_{j=1}^{N} U_j = \sum_{j=1}^{N} \frac{C_j}{T_j} \leq 1$

U è il **fattore di utilizzazione** del processore

> Il j-esimo termine della sommatoria $C_j/T_j = (C_j(H/T_j)) / H$ rappresenta la frazione dell'iperperiodo $H = mcm(T_1, T_2, ..., T_N)$ impiegata dal processo P_j

Schedulazione clock-driven

Schedulazione di tipo:

- offline
- guaranteed
- non preemptive

Non idonea in contesti che implicano dinamicità e flessibilità.

I parametri temporali sei processi si intendono noti a priori e non soggetti a variazioni runtime.

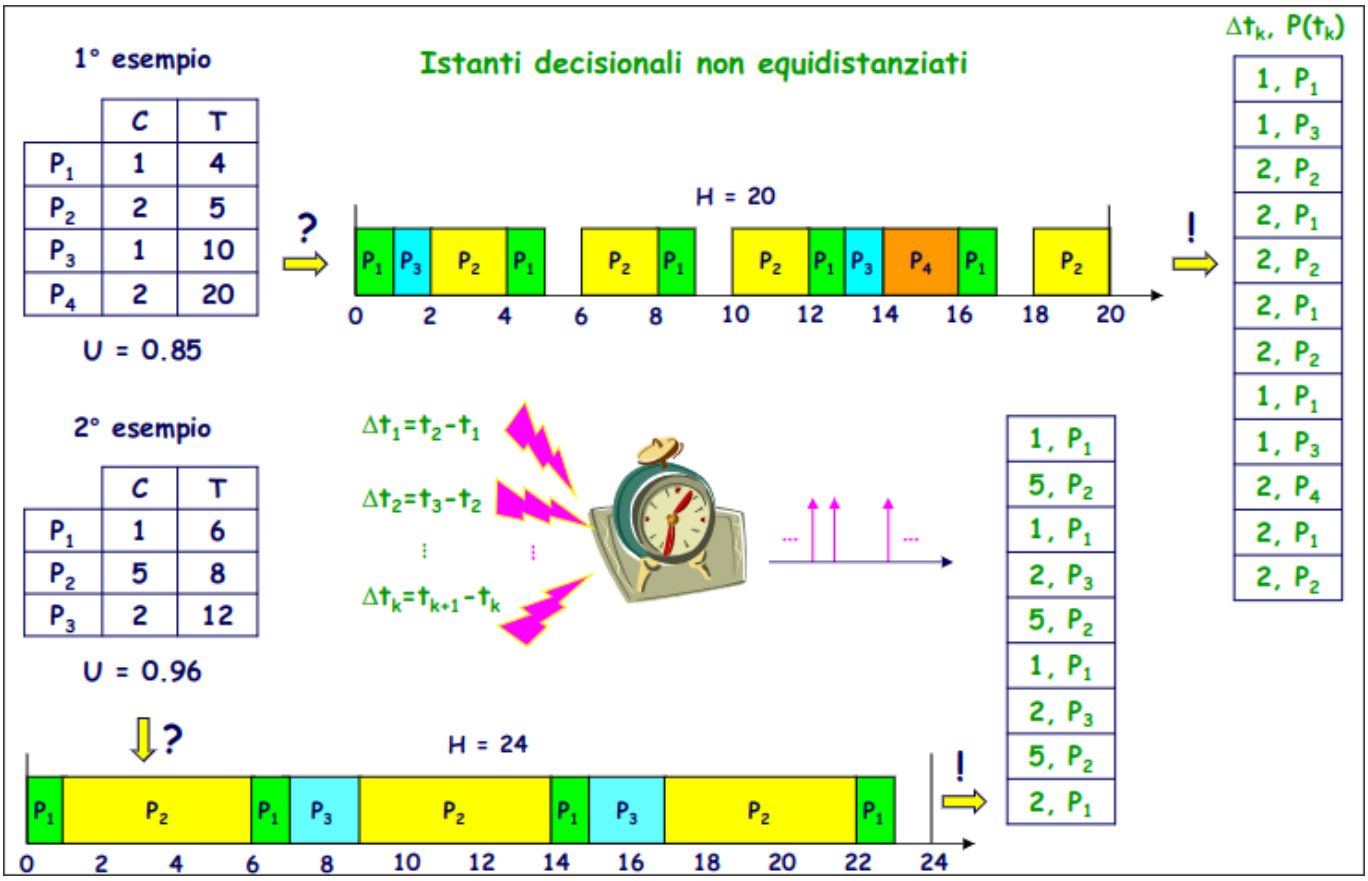
Tutti i vincoli temporali vengono soddisfatti a priori in sede di costruzione di un feasable schedule.

associate a processi NP-hard

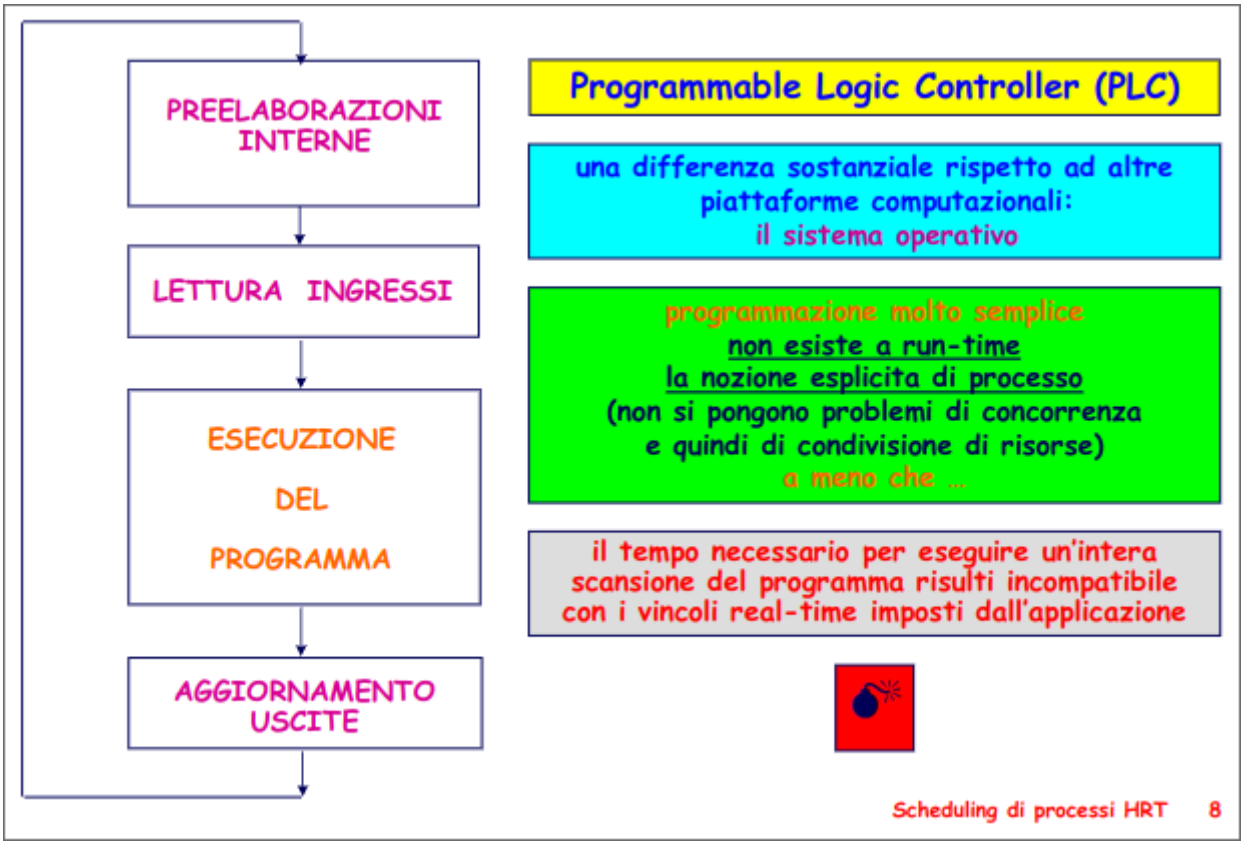
lo schedule viene fatto su un iperperiodo in istanti decisionali predefiniti.

> Ipotesi per un corretto funzionamento: **non job overrun**.

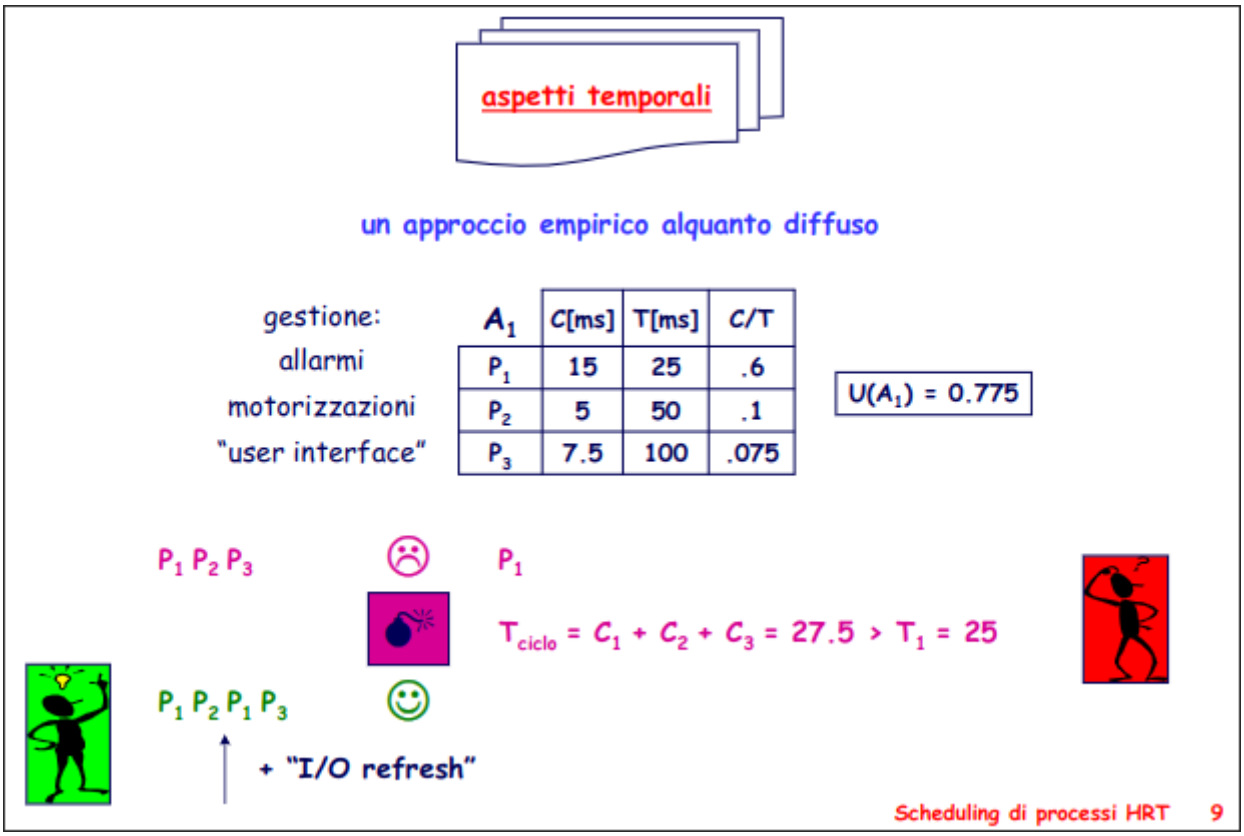
Timer-driven scheduling

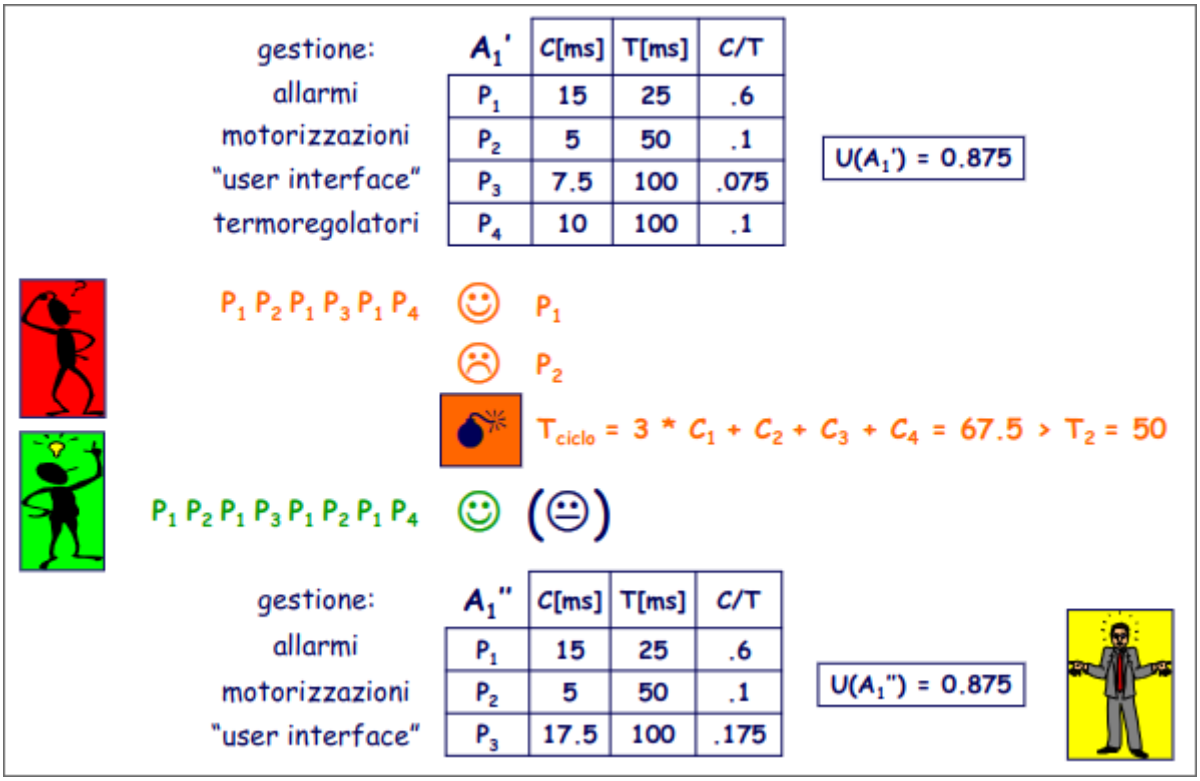


Ambiente di esecuzione

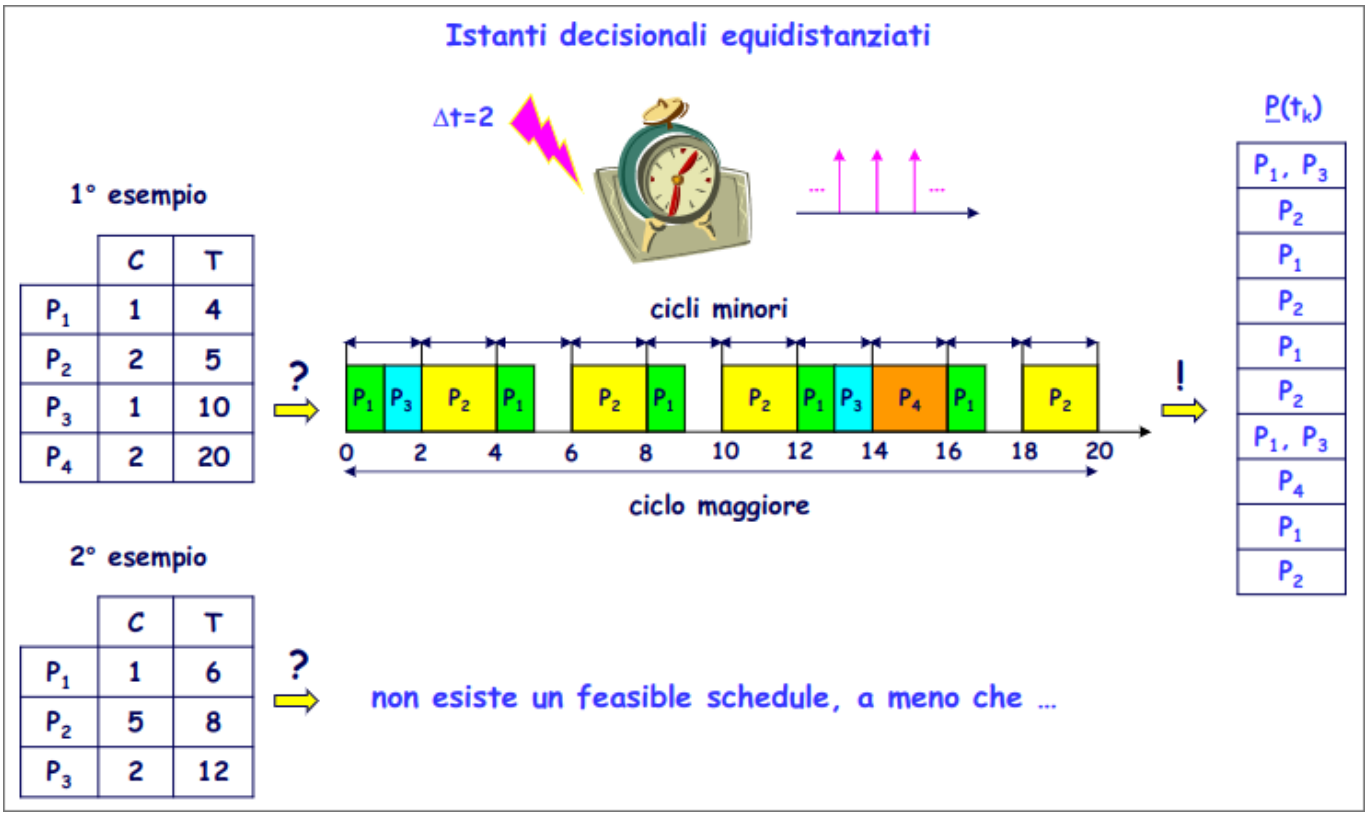


Sequenziale

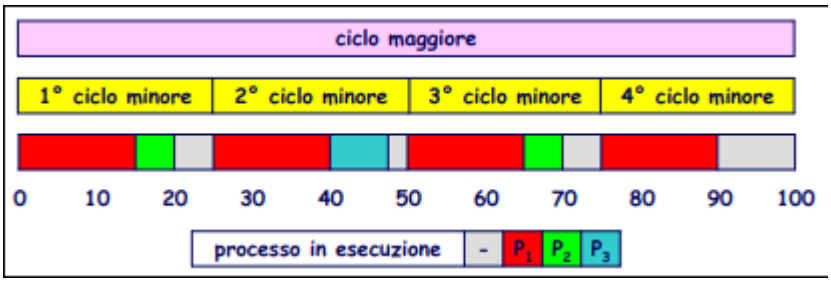




Cyclic Executive



Approccio Cyclic Executive



Supponiamo tre processi con periodi armonici 25, 50 e 100 ms. (20 50 100 non sono armonici)

Ciclio Maggiore: Periodo Maggiore **Ciclio Minore**: Periodo Minore

In questo caso avremo tre cicli minori

Un Task P_1 per ogni ciclo Minore Un Task P_2 per ogni due cicli Minore Un Tast P_3 in un solo ciclo Minore

PRO: Molto semplice **CONTRO**: Macchinoso con grandi differenze di periodo, poco applicabile

Si può aggiungere il job slicing (frammentazione di un task)

Approccio Barker - Shaw

Cliclo maggiore: $mcm(T_1, T_2, ..., T_N)$

Ciclio minore (Frame):

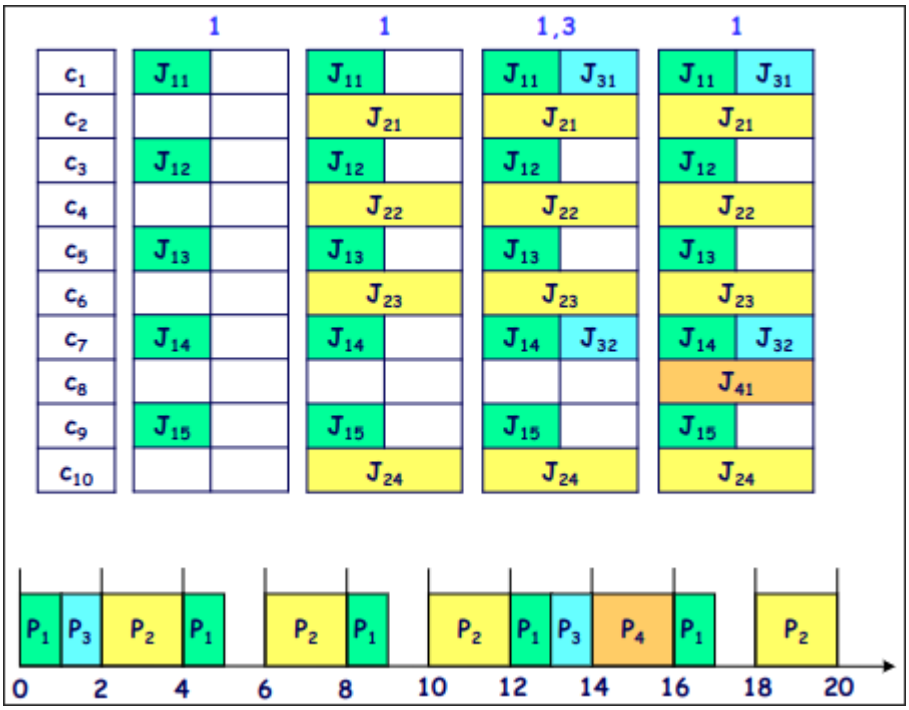
- $n mod m = 0$ un ciclo maggiore composto da un numero intero di cicli minori
- $m \geq c , \forall i$ no job preemption

- $m \leq T_i , \forall i$ in ogni ciclo maggiore vanno eseguiti tutti i task
- $2m - MCD(m,T_i) \leq T_i , \forall i \ | \ (T_i \bmod m) > 0$

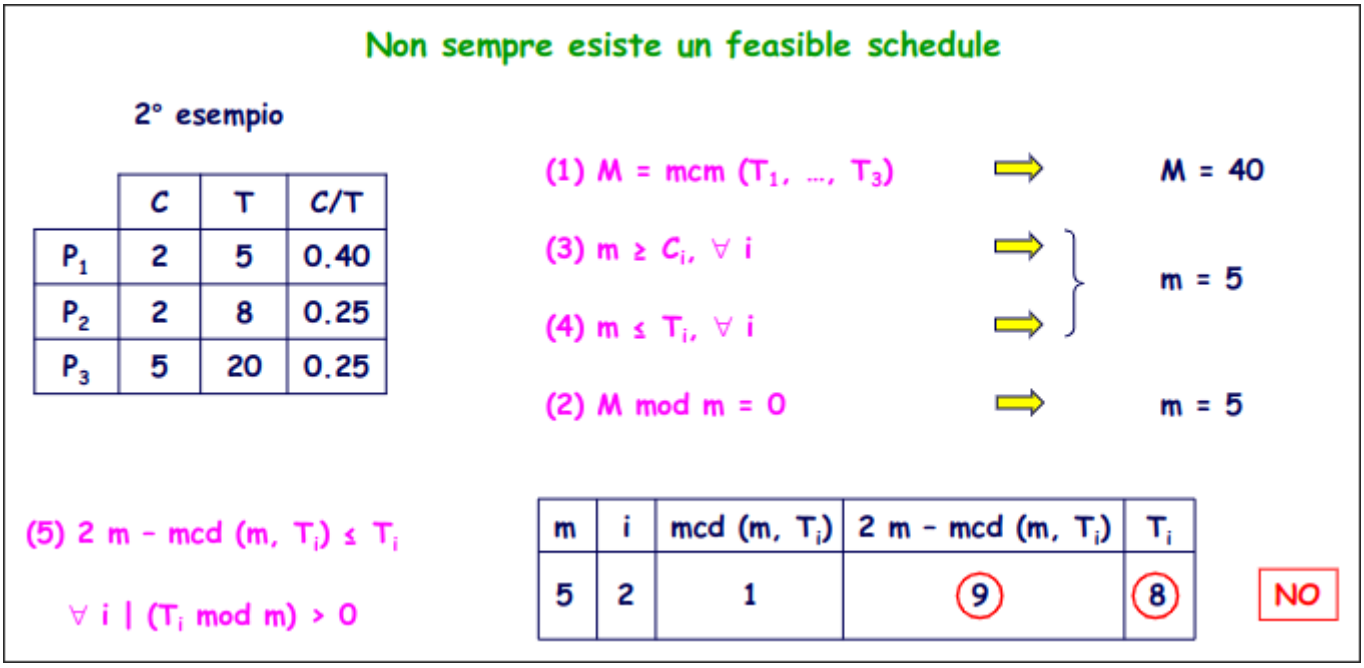
Costruzione di un feasable schedule

	C	T
P_1	1	4
P_2	2	5
P_3	1	10
P_4	2	20

M = 20, m = 2



Job Slicing: Si può adottare dividendo i job con un tempo di elaborazione più lungo, finche non si rispetta il vincolo 5.



> Dopo avere identificato n e m si applicano criteri euristici che possono portare a risultati differenti

Schedulazione Priority Driven

Ad ogni processo è associata una priorità statica o dinamica

Ogni processo può essere in stato:

- **Ready**: pronto per essere eseguito
- **Running**: in esecuzione
- **Idle**: in attesa di un evento

c'è preemption

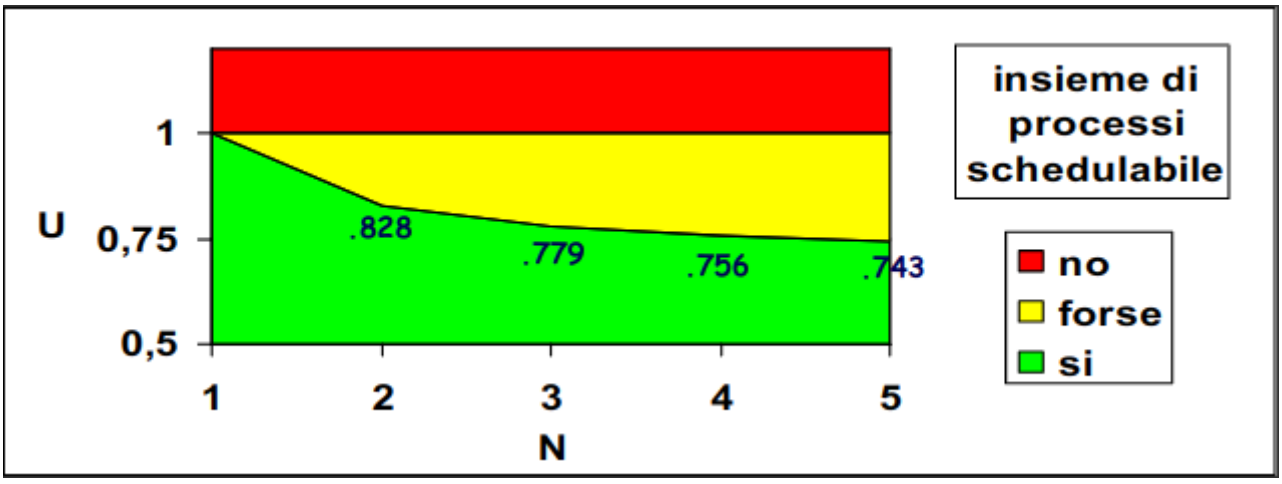
Algoritmo Rate Monotonic Priority Ordering (RMPO)

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

Algoritmo Ottimo: Un insieme id processi a priorità astatica se non è schedulabile con RMPO non è schedulabile

Test di schedulabilità LIU-LAYLAND

> Condizione sufficiente affinchè un insieme di N processi con RMPO: $U \leq U_{RMPO}(N) = N(2^{\frac{1}{N}} -1)$

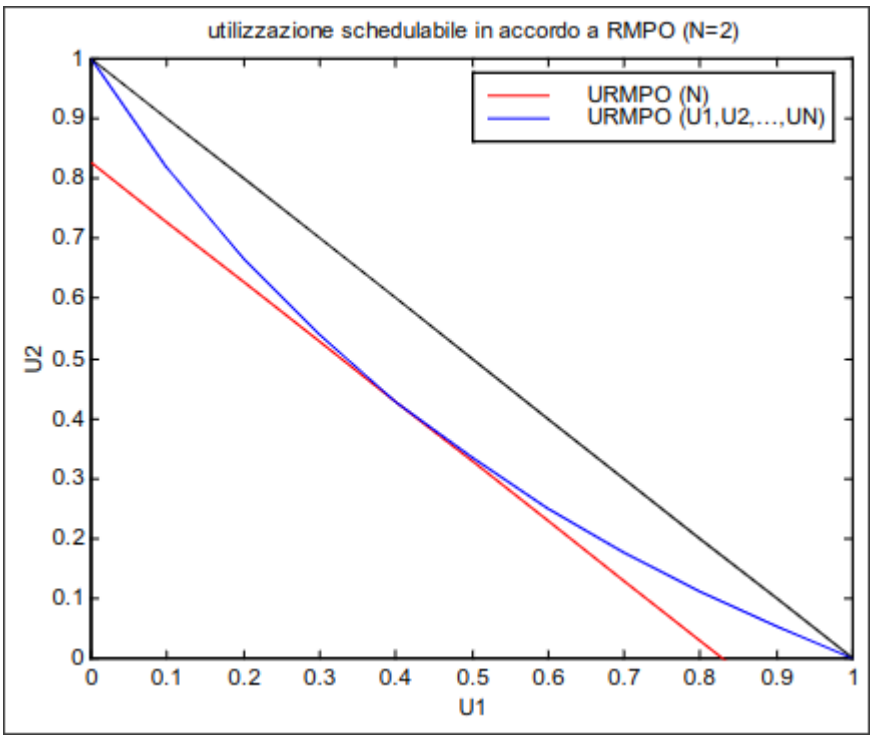


> $\lim_{N\rarr\inf} U_{RMPO}(N)=\ln 2 = 0.693$

Corollario

Test meno stringente del teorema (che fallisce spesso)

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



(Caso con due processi)

Quando i due fattori di utilizzazione sono simili il corollario da risultati simili al teorema.

Quando c'è differenza il corollario è meno stringente.

Test di Kuo - Mok

> Un insieme S di N processi P_i con $i = 1, 2, ..., N$ è schedulabile con RMPO se: $U \leq U_{RMPO}(K)$ essendo K il numero di sottoinsiemi disgiunti di processi semplicemente periodici in S.

Si Raggruppano i task con periodi armonici ottenendo dei nuovi task dove:

- $U_{nuovo} = U_x + U_y + ... + U_z$
- T_{nuovo} = min{T_x, T_y, ..., T_z}
- C_{nuovo} = U_{nuovo} * T_{nuovo}

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

Se il partizionamento non è univoco, allora optare per quello che ha fattori di utilizzazioni disuniformi.

Test di Burchard

> L'utilizzazione schedulabile dell'algoritmo RMPO è tanto maggiore quanto meno i periodi dei processi si discostano dalla relazione armonica

Per primo vanno calcolati: $X_j = \log_2(T_j) - \lfloor\log_2(T_j)\rfloor \quad \forall j$

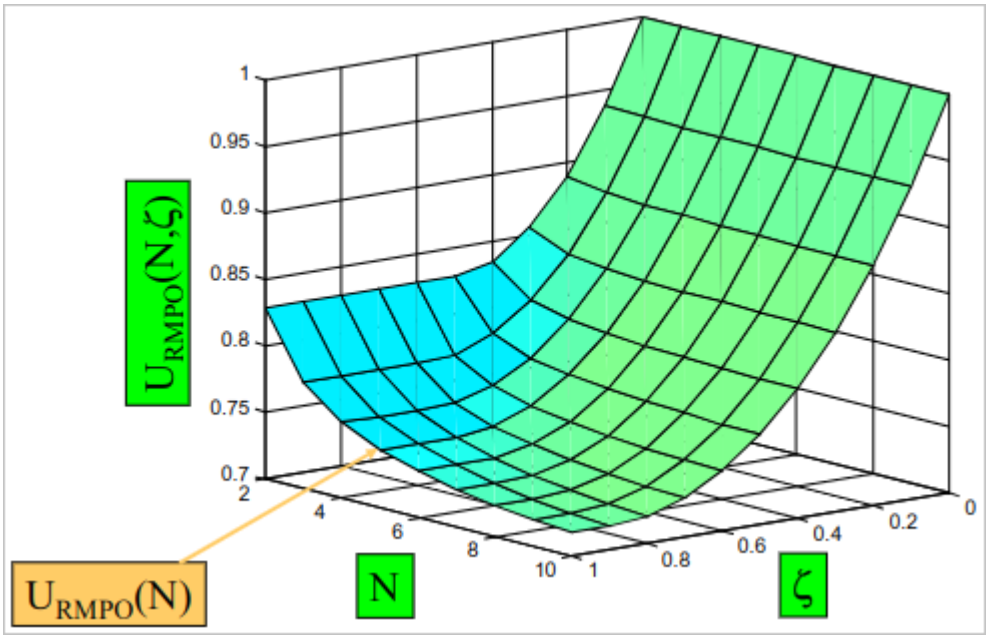
($\lfloor \log_2(T_j)\rfloor$ indica la parte intera del logaritmo in base 2 di T_j)

dopo di che si ottiene la **distorisione (ζ)** che indica di quanto i periodi si discostano dalla relazione armonica

$\zeta = {max(x_i)}{*1\leq j\leq N*} - {*min(x_j)*}{1\leq j \leq N}$

Ottenuto questo valore il coefficiente di utilizzazio massimo deve essere:

$$ \mathrm{\mathbf{U}_{RMPO}}(N, \zeta) = \begin{cases} (N-1) \left(2^{\zeta/(N-1)} - 1 \right) + 2^{1-\zeta} - 1 & \text{se } \zeta < 1 - \frac{1}{N}, \[10pt] N \left(2^{1/N} - 1 \right) & \text{se } \zeta \geq 1 - \frac{1}{N}. \end{cases} $$



dove $\zeta = 0$ indica che i periodi sono armonici quindi $U_{RMPO}=1$.

dove $\zeta = 1$ indica che i periodi sono tutti diversi quindi $U_{RMPO}=N(2^{1/N}-1)$, c'è molta distorsione.

Test di Han

> Un insieme S di N processi P_i con $i = 1, 2, ..., N$ è schedulabile con RMPO se ad esso corrsiponde un **insieme accelerato** S' di N processi P_i' con $i = 1, 2, ..., N$ semplicemente periodici con fattore di utilizzazione $U' = U_1' + U_2' + ... + U_N' \leq 1$

Se S' è schedulabile allora anche S è schedulabile.

Per creare l'insieme accelerato si prende il periodo minore e si mettono gli altri in relazione armonica con esso, se falliscono i test si prende il secondo e così via. Se quensto non funziona si può procedere in altri modi.

Si possono applicare più metodi in cascata per esempio l'insieme accelerato può essere sottoposto al corollario di Liu-Layland.

Analisi di schedulabilità di Audsley

Algoritmo di Audsley basato sul calcolo dei tempi di risposta.

La schedulabilità è garantita se il tempo di risposta di ogni processo non eccede la sua deadline.

$R_i=C_i+I_i(R_i)\leq T_i$ con $i = 1,2, ...,N$

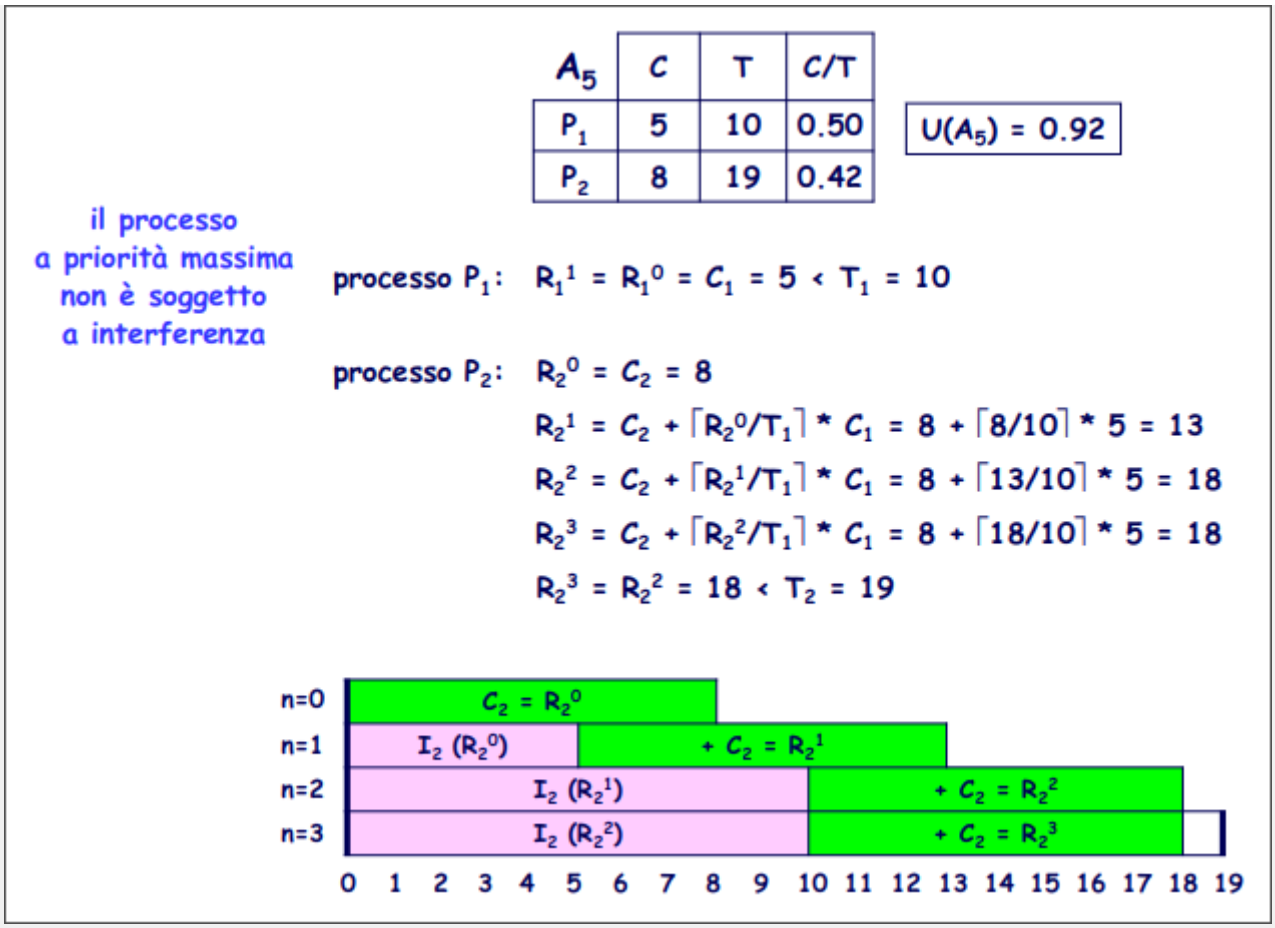
Dove I_i è l'interferenza sul tempo di risposta R_i del processo P_i dovuta ai processi con priorità maggiore.

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

R_i sarà:

$R_i^0 = C$, $R_i^n=C_i+I_i(R_i^{n-1})$ con $n=1,2,...$

> **Esempio**:



Alternativa all'algoritmo di Audsley

Meno calcoli, quindi test più veloci, ma meno efficace, solo condizioni sufficienti:

> $C_i + I_i(T_i) = C_i + \sum_{j|p_j>p_i} \lceil \frac{T_i}{T_j} \rceil C_j \leq T_i$ con $i = 1,2,...,N$

Processi Sporadici

> Tipicamente hanno una frequenza di esecuzione bassa ma una deadline stringente

Ogni processo sporadico P_i con $i = 1,2,...,N$ è caratterizzato da:

- T_i MIT (Minimum Interarrival Time) per i processi sporadici: tempo minimo tra due arrivi consecutivi di un processo sporadico, (nel caso di un processo periodico è il periodo).
- D_i Deadline, $\leq T_i$ nei processi sporadici, $= T_i$ nei processi periodici
- C_i Tempo di esecuzione massimo $\leq D_i$

Deadline Monotonic Priority Ordering (DMPO)

Ogni processo è associato a una priorità statica inversamente proporzionale alla sua deadline relativa:

$p_i \propto \frac{1}{D_i}$

Algoritmo ottimale, se un insieme di processi è schedulabile con un algoritmo a priorità statica allora è schedulabile con DMPO. Se non lo è con DMPO allora non lo è con nessun algoritmo a priorità statica.

Analisi di schedulabilità attraverso i tempi di risposta

Algoritmo di audsley, condizione **necessaria e sufficiente** affinché un insieme di N processi periodici e sporadici sia schedulabile con DMPO:

> $R_i = C_i + \sum_{j|p_j>p_i} \lceil \frac{R_i}{T_j} \rceil C_j \leq D_i$ con $i = 1,2,...,N$

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

> $C_i + \sum_{j|p_j>p_i} \lceil \frac{D_i}{T_j} \rceil C_j \leq D_i$ con $i = 1,2,...,N$

Quindi applico la formula (2) per tutti i task, per quelli che non hanno successo applico (1).

Altro test

Si basa sulla densità di utilizzazione.

Condizione sufficiente affinché un insieme di N processi periodici e sporadici sia schedulabile:

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

Test di Lehoczky

Sempre condizione sufficiente, definiamo $D_j = \delta_j * T_j$ con $j=1,2,...,N$

$$ \sum_{j=1}^{N} \frac{C_j}{T_j}= \mathbf{U}(N, \delta) = \begin{cases} N((2\delta)^{\frac{1}{N}}-1)+1-\delta & \text{se} 0.5\leq\delta\leq 1, \[10pt] \delta & \text{se} 0\leq\delta\leq 0.5 \end{cases} \delta = min(\delta_j) $$

Test di Utilizzazione Efficace dei Processi

f_j Fattore di utilizzazione efficace:

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

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

$$ f_j\leq U(N=\mod{H_n}+1, \delta=d_j) $$

Allora l'insieme di processi è schedulabile se tale condizione (sufficiente) è soddisfatta $\foall P_j$ con $j=1,2,...,N$.

Algoritmo Earliest Deadline First (EDF)

Ad ogni processo è associata una priorità dinamica inversamente proporzionale alla sua deadline relativa

Condizione necessaria e sufficiente affinché un insieme di N processi periodici e sporadici sia schedulabile con EDF:

> $U = \sum_{j=1}^{N} \frac{C_j}{T_j} \leq U_{EDF} = 1$

Condizione sufficiente affinché un insieme di N processi periodici e sporadici sia schedulabile con EDF:

> $\Delta = \sum_{j=1}^{N} \frac{C_j}{D_j} \leq 1$

Algoritmo Least Slack Time First (LST)

Ad ogni processo è associata una priorità dinamica inversamente proporzionale allo slack time

Non strict, non preemptive

strict, preemptive

Metascheduler

Considerando N task, il **Metascheduler** per gestire il release di tutti i task deve essere in grado di discriminare un tempo $T_{metascheduler}$:

> $T_{metascheduler} = mcd(T_1, ... , T_N)$

Il Metascheduler a sua volta è un task che va eseguito con periodo $T_{metascheduler}$, il cui compito è la gestione dell'esecuzione dei task.

Il sistema genera degli interrupt periodici, creando così il **Tick di sisitema**, fondamentale al Metascheduler per scandire il tempo.

Il Metascheduler è il task con priorità maggiore, quindi il sistema operativo cede sempre a lui le risorse, si occupa poi di avviare gli altri task.