Lezione R3

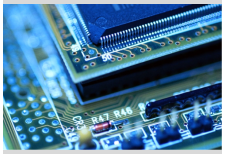
Schedulazione clock-driven

Sistemi embedded e real-time

2 ottobre 2020

Marco Cesati

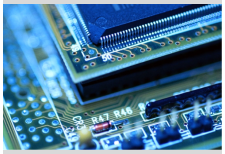
Dipartimento di Ingegneria Civile e Ingegneria Informatica
Università degli Studi di Roma Tor Vergata



Di cosa parliamo in questa lezione?

In questa lezione esaminiamo una classe di algoritmi di schedulazione dal comportamento deterministico e facili da validare

1. Algoritmi clock-driven
2. Cyclic executive
3. Suddivisione del tempo in frame
4. Gestione dei job aperiodici



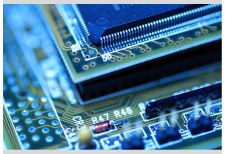
Tipologie di algoritmi per la schedulazione real-time

Esistono e vengano utilizzati un gran numero di differenti algoritmi per la schedulazione nei sistemi real-time

La maggior parte di essi possono essere ricondotti a tre grandi famiglie:

1. Algoritmi clock-driven
2. Algoritmi weighted round-robin
3. Algoritmi priority-driven

In questa lezione parliamo di algoritmi clock-driven



alla base dei sistemi di 20 anni fa (gestione risorse guidate dal clock)

Algoritmi clock-driven

Un algoritmo di schedulazione è detto essere *clock-driven* se le decisioni riguardanti i job da eseguire e gli intervalli di tempo in cui questi devono rimanere in esecuzione sono determinate in anticipo (off-line) e adottate in istanti di tempo predefiniti

tutto già previsto!

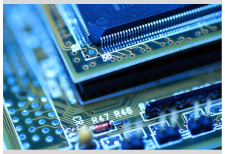
Tipicamente, in un sistema schedulato con un algoritmo clock-driven:

- Gli istanti in cui lo scheduler interviene sono fissati una volta per tutti
- L'insieme dei task periodici, con tutti i loro parametri funzionali ed i vincoli temporali, sono conosciuti e costanti, sono COSTANTI
- Una schedulazione opportuna può essere calcolata "off-line" dal progettista del sistema ed è seguita fedelmente dallo scheduler a "run-time"

"va bene per sempre"!!

Spesso i sistemi che adottano una schedulazione clock-driven utilizzano un componente hardware chiamato clock o timer in grado di generare interruzioni ad intervalli di tempo regolari

Schedulazione clock-driven
Marco Cesati



Schema della lezione
Algoritmi clock-driven
Cyclic executive
Job aperiodici soft r.-t.
Job aperiodici hard r.-t.

Perché sono utilizzati?

Quali sono i vantaggi più evidenti degli scheduler clock-driven?

L'algoritmo implementato dallo scheduler è molto semplice, quindi:

- lo scheduler è efficiente (ha un piccolo overhead)
- è facile validare il sistema nel caso di hard real-time

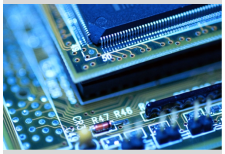
Predicibile

Qual è lo svantaggio più evidente degli scheduler clock-driven?

Lo scheduler è poco flessibile; ad esempio, è difficile gestire insiemi di task non periodici, oppure la creazione di nuovi task a run-time (non predicibili)

In passato, la maggior parte dei sistemi embedded hard real-time erano basati su uno scheduler di tipo **clock-driven**

La tendenza generale oggi è quella di adottare quando possibile scheduler **priority-driven**

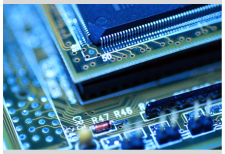


Perché sono utilizzati? (2)

- Gli scheduler di tipo clock-driven sono caratterizzati dal prendere le decisioni ad intervalli di tempo prefissati e costanti
  ↳ non sceglie quale job eseguire, segue ciò che impone il progettista
- Sono adatti per sistemi con alto grado di determinismo in cui i parametri di (quasi) tutti i job sono conosciuti a priori
- È possibile calcolare la migliore schedulazione possibile una volta per tutte (*off-line*): *schedulazione statica*
- Se applicata a task periodici, viene anche chiamata schedulazione *ciclica*

Per contrasto, gli algoritmi priority-driven:

- determinano la schedulazione ad ogni occorrenza di eventi dinamici come il completamento di un job o la creazione di un nuovo task
- sono quindi algoritmi *on-line* che effettuano una schedulazione *dinamica*



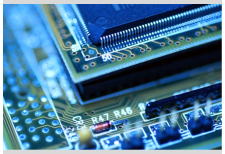
Modello a Task Periodici Ristretto

Per ragionare sugli algoritmi di schedulazione ciclica è utile fare riferimento ad una restrizione del modello a task periodici:

- Il numero n di task nel sistema è fissato
- I parametri di tutti i task periodici sono conosciuti a priori
- Ogni job può essere eseguito dal suo istante di rilascio (niente vincoli di precedenza o conflitti sulle risorse)
- Possono esistere job aperiodici con vincoli temporali soft e hard real-time

Notazioni per indicare i parametri di un task periodico T_i:

- (Φ_i, p_i, e_i, D_i): fase Φ_i, periodo p_i, tempo d'esecuzione e_i, scadenza relativa D_i
- (p_i, e_i, D_i): fase uguale a 0
- (p_i, e_i): fase 0, scadenza relativa uguale al periodo p_i



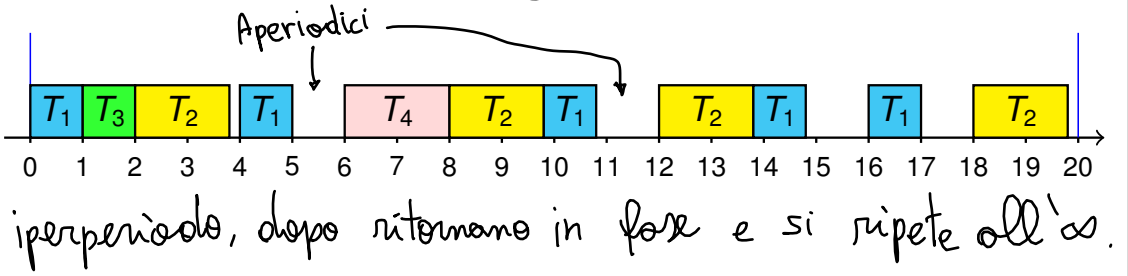
Esempio di schedulazione ciclica

Consideriamo un sistema con un processore e quattro task $T_1 = (4, 1)$, $T_2 = (5, 1.8)$, $T_3 = (20, 1)$ e $T_4 = (20, 2)$

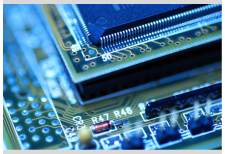
Come derivare una schedulazione fattibile?

- Lunghezza dell'iperperiodo: $\text{mcm}(4, 5, 20, 20) = 20$
- Troviamo una schedulazione fattibile in un iperperiodo
- Ripetiamo la schedulazione all'infinito

Una possibile soluzione:



Schedulazione clock-driven
Marco Cesati



Schema della lezione
Algoritmi clock-driven
Cyclic executive
Job aperiodici soft r.-t.
Job aperiodici hard r.-t.

Abbiamo visto:

Scheduler a tabella

Implementazione di uno scheduler clock-driven tramite tabella di voci $(t_k, T(t_k))$:

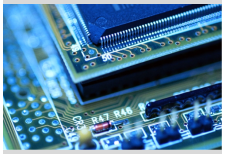
- t_k indica l'istante in cui una "decisione" è presa
- $T(t_k)$ indica il nome del job o task da eseguire oppure $\mathcal{I}$ se il processore è inutilizzato (*idle*)

La schedulazione precedente è rappresentata dalla tabella:

t_k	0	1	2	3.8	4	5	6	8	9.8
$T(t_k)$	T_1	T_3	T_2	$\mathcal{I}$	T_1	$\mathcal{I}$	T_4	T_2	T_1

10.8	12	13.8	14.8	16	17	18	19.8
$\mathcal{I}$	T_2	T_1	$\mathcal{I}$	T_1	$\mathcal{I}$	T_2	$\mathcal{I}$

- Job aperiodici soft real-time: schedulati negli intervalli "$\mathcal{I}$", ma interrotti se non completati entro il t_k successivo
- Job aperiodici hard real-time: non previsti (in questa versione)



Pseudo-codice per scheduler clock-driven

Procedura SCHEDULER:

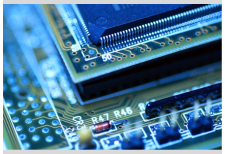
Input: Tabella di schedulazione $(t_0, T(t_0)), \ldots (t_{N-1}, T(t_{N-1}))$

$i = 0$, $k = 0$
imposta il timer a t_k (tempo assoluto)
ripeti:
    accetta interruzioni di timer
    interrompi un eventuale job aperiodico
    job corrente $J = T(t_k)$
    $i = i + 1$, $k = i \bmod N$
    imposta il timer a $\lfloor i/N \rfloor H + t_k$ (tempo assoluto)
    se $J = \mathcal{I}$
        attiva il primo job nella coda aperiodica
    altrimenti
        attiva il job J
    sospendi l'esecuzione dello scheduler

azione pesante!

Fine SCHEDULER

Schedulazione clock-driven
Marco Cesati



Schema della lezione
Algoritmi clock-driven
Cyclic executive
Job aperiodici soft r.-t.
Job aperiodici hard r.-t.

Schedulazioni cicliche strutturate

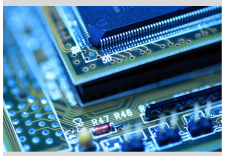
In generale è preferibile lavorare con schedulazioni che soddisfano certe proprietà strutturali, ad esempio:

- Attivazione dello scheduler ad intervalli regolari
- Distribuzione regolare degli intervalli "$\mathcal{I}$" (processore idle) nell'iperperiodo

Che vantaggi portano queste proprietà?

- Lo scheduler può essere attivato da un dispositivo hardware che genera interruzioni periodiche (ad es. il PIT, Programmable Interval Timer)
- I job aperiodici possono essere eseguiti in modo regolare in corrispondenza degli intervalli "$\mathcal{I}$"
- Possibilità di monitorare e/o forzare il rispetto dei vincoli temporali in caso di job che allungano la loro esecuzione

Una procedura che implementa un algoritmo di schedulazione ciclica "strutturata" è chiamata *cyclic executive*



Frame

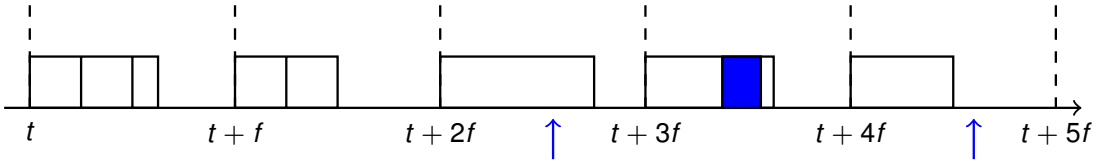
Gli istanti di tempo in cui uno scheduler ciclico strutturato prende decisioni partizionano la linea temporale in intervalli regolari chiamati *frame*

- La lunghezza f dei frame è prefissata
- In ogni frame è definita una lista di job da eseguire in sequenza (*blocco di schedulazione*)
- All'interno dei frame non si possono interrompere i job: se un job inizia in un frame, deve terminare entro lo stesso frame
- La fase di ogni task periodico è un multiplo intero non negativo della lunghezza del frame:

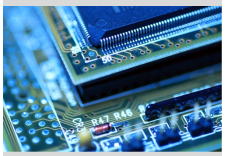
$$\forall i \in \{1, \ldots, n\}, \exists k \in \mathbb{N} : \Phi_i = k \times f$$

istante di rilascio del 1° job

se non fosse così, "taglierei il frame"!



Solo 1° rilascio allineato all'inizio del frame. Se Job arriva in "r" lo metto in "t+3f". Se arriva in "t+3f" RIMANE LÌ



Se job finisce a "t+4f" scheduler non lo sa (si sveglia quando è concluso), lo vede se termina in "t+4f+x = d"

Vincoli sulla dimensione dei frame (deve dimostrarle!)

(1) Poiché non è possibile interrompere un job all'interno di un frame, il frame deve essere abbastanza lungo da garantire la completa esecuzione di ciascun job:

$$f \geq \max\{e_1, \ldots, e_n\}$$

Job eseguito nell'intero frame

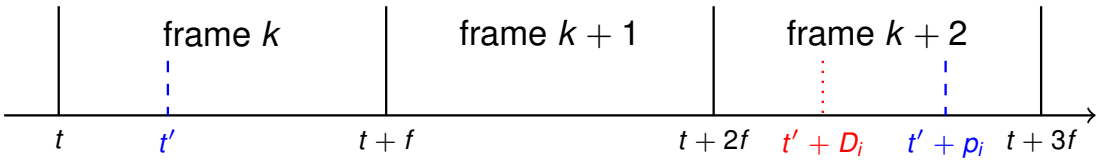
(2) La dimensione del frame deve dividere la lunghezza dell'iperperiodo H:

$$H/f - \lfloor H/f \rfloor = 0 \qquad \frac{f}{H} = \text{intero}$$

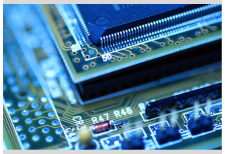
Condizione sufficiente è che f divida il periodo p_i di almeno uno dei task. (sennò iperperiodo in mezzo al frame)

(3) Il frame deve essere abbastanza piccolo così che tra l'istante di rilascio e la scadenza di ogni job ci sia sempre almeno un frame; condizione sufficiente:

$$2 \cdot f - \gcd(p_i, f) \leq D_i, \quad \forall i \in \{1, \ldots, n\} \quad \forall \text{ task}$$

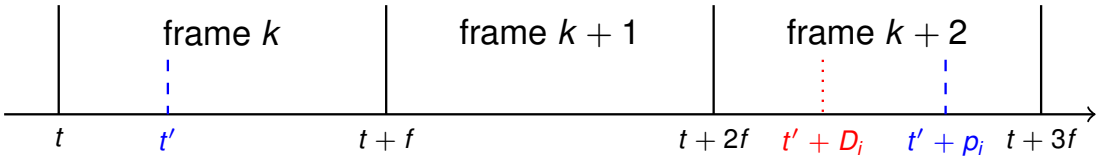


t' lo vedrò solo a "t+f", da li inizio a schedularlo



Vincoli sulla dimensione del frame (2)

inizio frame — inizio Frame



- Frame inizianti a t, $t+f$, $t+2f$, $t+3f$
- Rilasci a $t' \geq t$ e $t'+p_i$, scadenza $t'+D_i$ (di ri')
- $t' = \Phi_i + h' \cdot p_i = h \cdot f + h' \cdot p_i, \quad t = h'' \cdot f \quad (h, h', h'' \in \mathbb{N})$
- Se $g = \gcd(p_i, f)$ (mcd): $t' - t \stackrel{?}{=} g \cdot \left(\frac{h \cdot f}{g} + \frac{h' \cdot p_i}{g} - \frac{h'' \cdot f}{g}\right) \doteq g \cdot h'''$ (interi!)

t = fase job + periodi job precedenti a me.

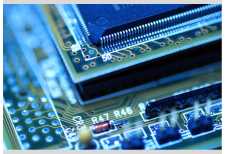
Caso 1: $t' > t$

- $h''' \in \mathbb{N}, \; t' - t > 0 \quad \Rightarrow t' - t \geq g$ (se $h'' > 0$ allora $h''' \neq 0$, minimo '1')
- $2f - g \leq D_i \Rightarrow \; 2f - (t' - t) \leq D_i \; \Rightarrow \; \boxed{t + 2f \leq t' + D_i}$ (sta oltre il frame)

Caso 2: $t' = t$

- $2f - \gcd(p_i, f) \leq D_i \quad \Rightarrow \quad f \leq D_i \quad \Rightarrow \quad \boxed{t + f \leq t' + D_i}$

($\gcd \leq f$)



Esempio di scelta della dimensione del frame

Consideriamo un sistema con quattro task

$T_1 = (4, 1)$, $T_2 = (5, 1.8)$, $T_3 = (20, 1)$ e $T_4 = (20, 2)$

Come scegliere la dimensione del frame?

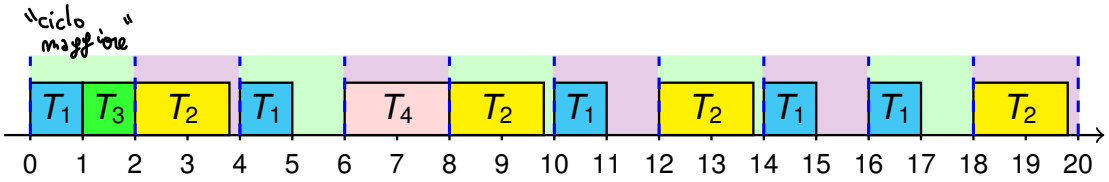
(1) $f \geq \max\{1, 1.8, 1, 2\} = 2$

(2) $H = 20$, quindi $f \in \{1, 2, 4, 5, 10, 20\}$

f divide 20 e non 1

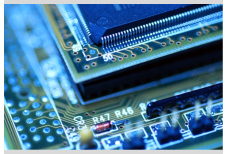
(3) $2f - \gcd(4, f) \leq 4, \quad 2f - \gcd(5, f) \leq 5,$
$2f - \gcd(20, f) \leq 20$, quindi $f \leq 2$

Risultato: dimensione del frame $f = 2$



Non è detto che ci si riesca

I gruppi di frame consecutivi (a cominciare dal primo frame) di lunghezza pari ad un iperperiodo sono detti *cicli maggiori*; ciascun frame è anche detto *ciclo minore*



Esempio di scelta della dimensione del frame (2)

Consideriamo un sistema con tre task

$T_1 = (4, 1)$, $T_2 = (5, 2, 7)$, $T_3 = (20, 5)$ esoso, se lo spezzassi?

Come scegliere la dimensione del frame?

(1) $f \geq \max\{1, 2, 5\} = 5$

(2) $H = 20$, quindi $f \in \{1, 2, 4, 5, 10, 20\}$

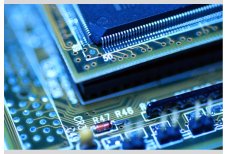
(3) $2f - \gcd(4, f) \leq 4, \quad 2f - \gcd(5, f) \leq 7,$
$2f - \gcd(20, f) \leq 20$, quindi $f \leq 4$

Risultato: non esiste una dimensione adatta!

Come si può rimediare?

Il problema è dovuto al vincolo (1): possiamo spezzare uno o più job in modo da ridurre i tempi di esecuzione

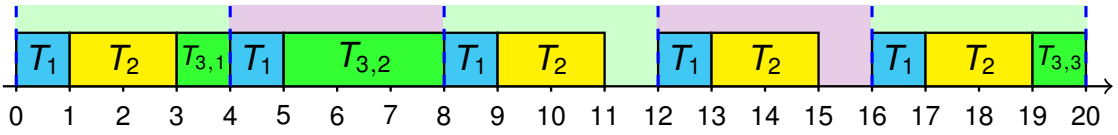
Ovviamente, la possibilità di farlo realmente dipende dalla natura dei job troppo lunghi



Frammentazione dei job

Nell'esempio precedente suddividiamo ogni job di $T_3 = (20, 5)$ in tre frammenti: $T_{3,1} = (20, 1)$, $T_{3,2} = (20, 3)$, $T_{3,3} = (20, 1)$

È quindi possibile scegliere come dimensione del frame $f = 4$



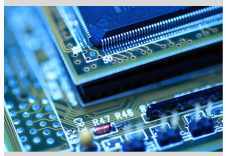
L'insieme di job frammentati è equivalente ai job di cinque task periodici $(4, 1)$, $(5, 2, 7)$, $(20, 1)$, $(20, 3)$, $(20, 1)$*?* ***No!***

Esistono vincoli di precedenza tra i frammenti!

In generale, per costruire una schedulazione ciclica dobbiamo:

- scegliere una dimensione del frame
- frammentare i job
- piazzare i frammenti nei frame (*blocchi di schedulazione*)

Ma le scelte non sono indipendenti tra loro!



Come gestire i task non armonici?

Consideriamo i task $T_1 = (3, 1)$, $T_2 = (7, 3)$ e $T_3 = (25, 3)$

L'iperperiodo è $H = 3 \cdot 7 \cdot 25 = 525$, l'unica dimensione ammissibile per il frame è 3 $\Longrightarrow$ il ciclo maggiore ha 175 frame

Svantaggio principale: spreco di memoria per la tabella contenente i blocchi di schedulazione

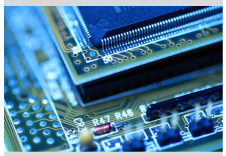
Come gestire la situazione?

Abbassando i periodi di alcuni task nei requisiti di progetto!

Ad esempio: $T_1 = (3, 1)$, $T_2 = (6, 3)$, $T_3 = (24, 3)$ $\Longrightarrow$ $H = 24$ ed il ciclo maggiore ha 8 frame

Abbassare il periodo di un task è come alzare uno stipendio: nessuno si lamenterà, ma debbono esserci soldi per tutti...

Nell'esempio U aumenta di $3/6 - 3/7 + 3/24 - 3/25 = 7.6\%$



Come gestire periodi non interi?

Se i periodi dei task non sono interi non è possibile applicare direttamente le formule per i vincoli sulla dimensione del frame

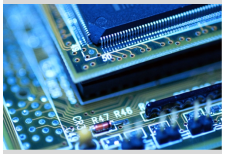
Ad esempio: $T_1 = (1.5, 0.5)$, $T_2 = (2.25, 0.25)$, $T_3 = (3, 0.75)$

Come si può risolvere il problema? Il tempo è virtuale, posso farlo!

Moltiplicando tutti i tempi per un fattore costante in modo da ottenere periodi interi

Ad es. moltiplicando per 4: $T_1' = (6, 2)$, $T_2' = (9, 1)$, $T_3' = (12, 3)$

Risolvendo si ottiene ad es. $f' = 6$, ossia un frame di dimensione $f = 1.5$ per il sistema originale



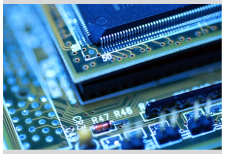
Pseudo-codice per cyclic executive

Procedura CYCLIC_EXECUTIVE:

Input: Blocchi di schedulazione $L(0), L(1), \ldots, L(F-1)$;
Code di job aperiodici

$t = 0$, $k = 0$
ripeti:
accetta interruzione di clock al tempo $t \cdot f$ (assoluto)
blocco di schedulazione corrente $B = L(k)$
$t = t + 1$, $k = t \bmod F$
gestisci il caso di mancata conclusione dell'ultimo job
gestisci il caso di job (o frammento) in B non eseguibile
sveglia il server dei task periodici per eseguire B
sospendi l'esecuzione fino alla conclusione del server
ripeti finché la coda di job aperiodici è non vuota:
attiva il job in cima alla coda
sospendi l'esecuzione fino alla conclusione del job
rimuovi il job dalla coda
sospendi l'esecuzione (fino alla successiva interr. di clock)

Fine CYCLIC_EXECUTIVE



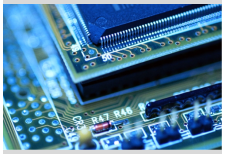
Schedulazione di job aperiodici soft real-time

In un cyclic executive riveste molta importanza la gestione dei job aperiodici soft real-time:

- Sono eseguiti "in background" quando il processore non è occupato dai task periodici
- La loro esecuzione può essere ritardata
- Sono tipicamente attivati in conseguenza di eventi esterni
- Più rapido è il loro completamento, migliore appare la reattività del sistema rispetto ai segnali esterni

Minimizzare i tempi di risposta dei job aperiodici soft real-time è un obiettivo di progetto degli algoritmi di schedulazione

Come è possibile ottenere questo risultato se in ogni frame dobbiamo comunque gestire i task periodici?



Slack stealing

Una tecnica per migliorare i tempi di risposta dei job aperiodici soft real-time chiamata *slack stealing* è stata proposta da Lehoczky e Ramos-Thuel nel 1992

Per ogni frame k, sia x_k l'ammontare di tempo già allocato, e sia $f - x_k$ lo slack (margine di tempo ancora disponibile)

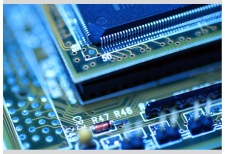
In ogni frame, lo scheduler può eseguire i job aperiodici **prima** di quelli periodici se lo slack non è nullo (basta che i RT finiscono prima del frame)

L'implementazione richiede:

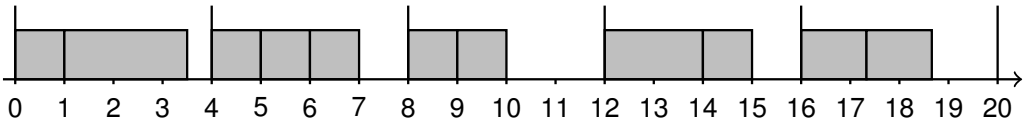
- di calcolare la quantità di slack in ogni frame lasciata libera dai job periodici
- di tenere traccia della quantità di slack consumata dai job aperiodici durante la loro esecuzione

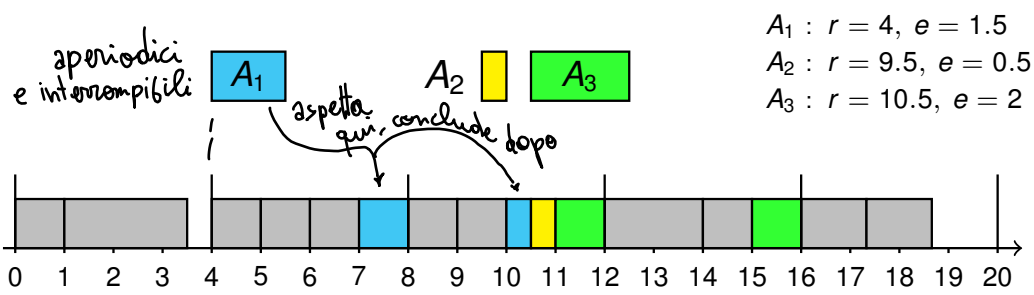
  si può utilizzare ad esempio un interval timer impostato all'inizio del frame con il valore dello slack disponibile

È possibile utilizzare lo slack stealing anche con algoritmi priority-driven, ma è molto più complicato

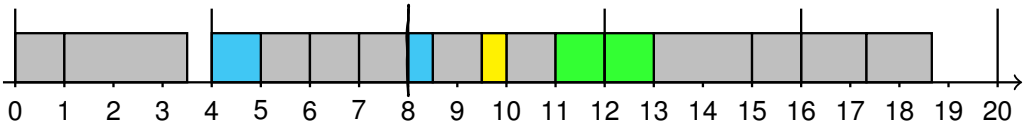


Esempio di applicazione di slack stealing

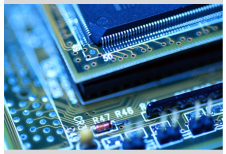




Tempi di risposta senza s.s.: A_1: 6.5, A_2: 1.5, A_3: 5.5



Tempi di risposta con s.s.: A_1: 4.5, A_2: 0.5, A_3: 2.5



Schedulazione di job aperiodici hard real-time

I job aperiodici hard real-time vengono rilasciati ad istanti di tempo arbitrari, e se ne conoscono i parametri solo dopo il rilascio

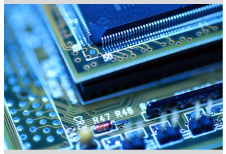
A differenza di quelli soft real-time, i job aperiodici hard real-time non possono mancare la loro scadenza

Per ogni schedulazione ciclica è sempre possibile trovare un job aperiodico hard real-time che non può essere completato entro la scadenza a meno di ritardare uno o più job periodici

Anche i task periodici sono hard real-time: come schedulare insieme i job periodici e quelli aperiodici?

Per ogni nuovo job aperiodico hard real-time che viene rilasciato:

- Il cyclic executive invoca un *test di accettazione*
- se il test dimostra che il nuovo job può essere completato entro la scadenza senza danneggiare gli altri job hard real-time, viene accettato e schedulato
- altrimenti il job viene rifiutato



Schedulazione EDF per job aperiodici hard real-time

Un modo efficiente di gestire i job aperiodici hard real-time è quello di utilizzare l'algoritmo EDF: ha la precedenza il job con scadenza più vicina

La procedura cyclic executive utilizza due code di job aperiodici hard real-time ordinate secondo l'istante di scadenza:

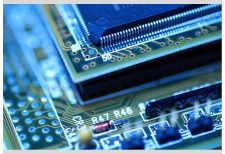
- Una coda EDF per i job rilasciati e non ancora accettati
- Una coda EDF per i job accettati, prendo in cima a questa coda.

L'accettazione dei job e la schedulazione della loro esecuzione vengono sempre effettuate all'inizio di un frame

L'algoritmo EDF è ottimale, ossia sempre in grado di trovare una schedulazione fattibile per i job aperiodici hard real-time se questa esiste?

È ottimale solo nell'ambito dei vincoli imposti dai frame

Esempio: un job aperiodico potrebbe essere schedulabile solo se fosse eseguito non appena rilasciato in mezzo ad un frame



Test di accettazione dei job aperiodici hard real-time

Quali sono i principali limiti del test di accettazione?

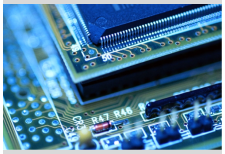
Si deve assumere che tutti i job aperiodici hard real-time: LI DEVO CONOSCERE

- possono essere suddivisi in gruppi di cui si conosce in anticipo il tempo d'esecuzione (massimo)
- sono interrompibili, quindi la loro esecuzione può essere suddivisa su più frame

In cosa consiste il test di accettazione?

Si basa sul calcolo dello slack disponibile nei frame interamente compresi tra l'istante di rilascio e la scadenza

- Un job aperiodico hard real-time S viene rilasciato con scadenza d e tempo d'esecuzione e
- Il frame successivo all'istante di rilascio ha numero t ($1 \leq t \leq F$) all'interno del ciclo maggiore j
- Il frame precedente a quello in cui cade la scadenza ha numero ℓ ($1 \leq \ell \leq F$) nel ciclo maggiore j'
- Il cyclic executive esegue il test all'inizio del frame t



Test di accettazione dei job aperiodici hard real-time (2)

- La quantità di slack $\sigma(i, h)$ lasciata libera dai job periodici è precalcolata per ogni ciclo maggiore ($1 \leq i, h \leq F$)
- Su più cicli maggiori la quantità di slack totale è:

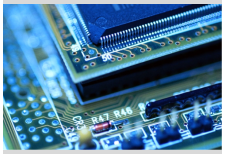
  $$\sigma(i + j\,F, h + j'\,F) = \sigma(i, F) + (j' - j - 1) \cdot \sigma(1, F) + \sigma(1, h)$$

- Per ogni job aperiodico hard real-time S_k già accettato all'inizio del frame t si conosce la scadenza d_k, il lavoro ancora da svolgere $e_k - \xi_k$, lo slack rimanente σ_k
- La quantità totale di slack disponibile tra i frame t e ℓ:

  $$\sigma_c(t, \ell) = \sigma(t, \ell) - \sum_{d_k \leq d} (e_k - \xi_k)$$

  deve essere $\geq e$, altrimenti S viene rifiutato
- In caso di accettazione, per ogni job aperiodico hard real-time con scadenza oltre d lo slack rimanente deve essere diminuito di e: S viene rifiutato anche se $\sigma_k - e < 0$ per qualche k
- Se S è accettato, il suo slack è inizialmente

  $$\sigma = \sigma_c(t, \ell) - e$$



Gestione dei job aperiodici hard r.-t. nel cyclic executive

Procedura CYCLIC_EXECUTIVE:

Input: ... code di job ap. hard r.-t. non accettati e accettati

⋮

gestisci il caso di job (o frammento) in B non eseguibile

ripeti finché coda di job ap. hard r.-t. non accettati è non vuota:
    preleva il job in cima alla coda
    esegui il test di accettazione sul job
    se il job è eseguibile, inserisci in coda job accettati
    altrimenti segnala "job rifiutato"

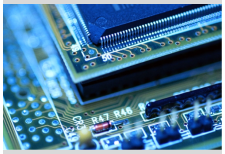
sveglia il server dei task periodici per eseguire B
sospendi l'esecuzione fino alla conclusione del server

Come prima + logica code

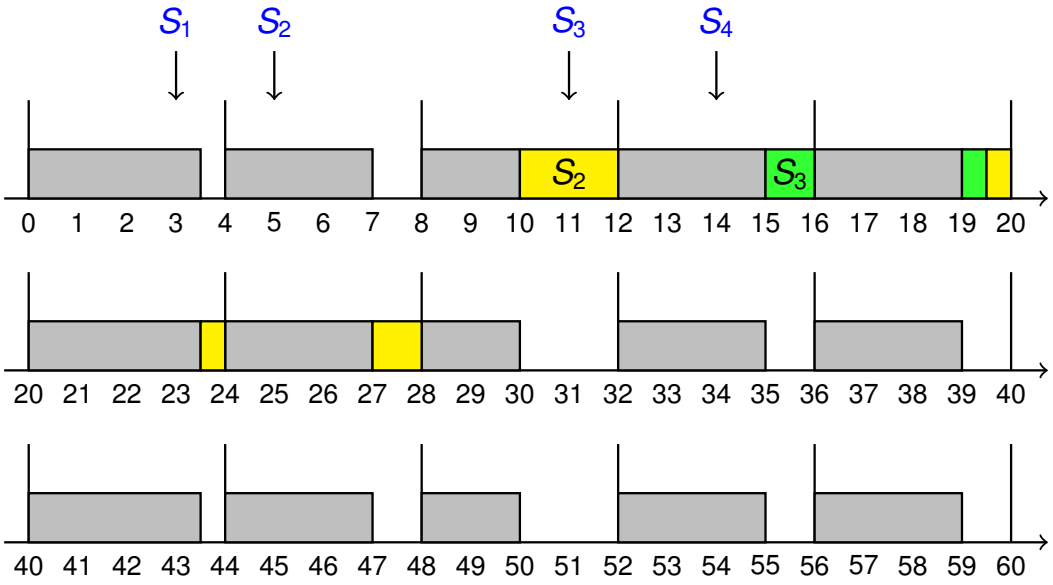
ripeti finché la coda di job ap. hard r.-t. accettati è non vuota:
    sveglia il job in cima alla coda
    sospendi l'esecuzione fino alla conclusione del job
    rimuovi il job dalla coda

ripeti finché la coda di job ap. soft r.-t. è non vuota:

⋮



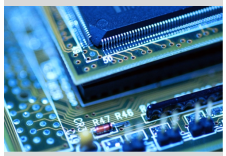
Esempio di schedulazione EDF di job aperiodico hard r.-t.



Il primo elemento di sigma è il numero del frame successivo al frame in cui arriva il job aperiodico. Si parte da 1.
Il secondo elemento di sigma è il numero del frame precedente al frame in cui c'è la scadenza del job.

	tempo	scadenza	durata			
S_1	$r_1=3$	$d_1=17$	$e_1=4.5$	$\sigma_c(2,4)=4 < e_1$	RIF	
S_2	$r_2=5$	$d_2=29$	$e_2=4.0$	$\sigma_c(3,7)=5.5 > e_2$	ACC	$\sigma_2=1.5 = 5,5-4$
S_3	$r_3=11$	$d_3=22$	$e_3=1.5$	$\sigma_c(4,5)=2$ $\sigma_2 \geq e_3$	ACC	$\sigma_2=0, \sigma_3=0.5$
S_4	$r_4=14$	$d_4=44$	$e_4=5.0$	$\sigma_c(5,11)=4.5$	RIF	

Scambio S_3 e S_2



Gestione delle violazioni delle scadenze

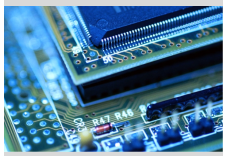
Un job hard real-time completa sempre entro la sua scadenza a meno di malfunzionamenti dell'hardware, bug nel software, o difetti di progettazione del sistema real-time

Uno scheduler progettato in modo robusto all'inizio del frame successivo alla scadenza controlla che il job sia effettivamente terminato

In caso contrario lo scheduler cerca di porre rimedio: se fallisco scadenza HRT?

- Elimina completamente il job non terminato
- Interrompe il job ed alloca la parte di esecuzione restante come job aperiodico (magari è utile comunque)
- Continua l'esecuzione del job, allungando il frame contenente la scadenza e ritardando così tutti i frame successivi

L'azione di recupero più appropriata dipende ovviamente dalla natura del job e degli altri task del sistema, dipende dal contesto



Vantaggi e svantaggi degli scheduler clock-driven

- Sono concettualmente semplici e facili da validare
- Non è necessario controllare l'accesso alle risorse condivise
- Non è necessario sincronizzare tra loro i job
- Scegliendo opportunamente la durata dei frame è possibile minimizzare l'overhead dello scheduler
- Possono essere ancora semplificati assumendo che gli eventi esterni si verificano in sincronia con i frame
- Gli istanti di rilascio dei job devono essere prefissati
- Tutte le possibili configurazioni del carico devono essere previste in anticipo
- Non efficienti per sistemi con molti job aperiodici

per questo usa gli altri!