Contents

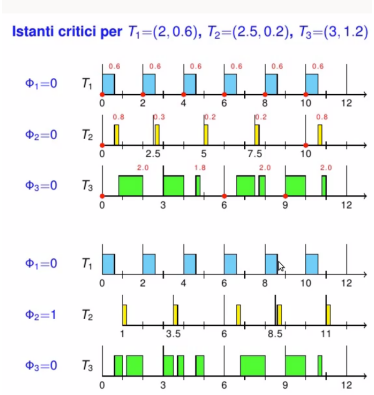
1 Schedulabilità di algoritmi a priorità fissa

Algoritmi a priorità dinamica, come EDF, sono ottimali (sotto determinate condizioni): se $\exists$ schedulazione fattibile $\Rightarrow$ anche EDF trova schedulazione.
Nessun algoritmo X a priorità fissa può avere un fatto di utilizzazione $U_X = 1$, deve per forza essere < 1.
Inoltre RM è ottimale (in senso assoluto, ovvero può raggiungere $U = 1$) per sistemi armonici con scadenze implicite.
In questa condizione RM è tanto buono quanto EDF.
DM è ottimale tra gli algoritmi a priorità fissa, ma non in senso assoluto: se $\exists$ algoritmo a priorità fissa che trova una schedulazione fattibile per un insieme di task, allora lo fa anche DM. Questo mi fa capire assegnare priorità fisse ai task,in modo arbitrario, non fa guadagnare nulla rispetto ad assegnarle con un parametro come la scadenza relativa. Algoritmo è altrettanto buono, se non più buono di algoritmi che fissano le scadenze in modo soggettivo, posso realizzare sistemi di task basati su parametri oggettivi e non soggettivi.
Corollario: RM è ottimale tra gli algoritmi a priorità fissa per sistemi ti task con scadenza proporzionale al periodo.

Mi pongo un problema generale: se ho sys di task generale ed un algoritmo di schedulazione a priorità fissa, come faccio a verificare il sistema, ovvero a certificare che l'algoritmo produrrà sempre una schedulazione valida?

1.1 Istanti critici

Istanti critici: suppongo che nel sys di task tutti i job abbiano un tempo di risposta piccolo, ovvero ogni job termina prima del rilascio del job successivo del task ⇒ ogni job viene rilasciato in un periodo e si conclude entro quel periodo (job potrebbe non rispettare la scadenza, se questa è minore del periodo).
L'istante critico è il momento in cui il rilascio del job comporta il massimo tempo di risposta possibile per quel job.
Se almeno un job T_i non rispetta la scadenza relativa, l'istante critico è un momento in cui il rilascio di un job provoca il mancato rispetto della scadenza di quel job.
Io voglio verificare che tutti i job rispettino le scadenze, sottigliezza della definizione è irrilevante dal punto di vista critico.
Teorema: Se ho un sistema di task a priorità fissa e tempi di risposta piccoli, l'istante in cui uno dei job di T_i viene rilasciato contemporaneamente ai job di tutti i task con priorità maggiore di T_i è l'istante critico di T_i.
Teorema non da condizione necessaria e sufficiente, ma solo sufficiente: se capita tale condizione ⇒ ho un istante critico, ma potrei averne altri.
esempio : T_1=(2, 0.6) T_2=(2.5,0.2), T_3=(3,1.2).
T_1 ha la priorità massima: tutti i multipli di 2 sono istanti critici.
T_2 ha istanti critici 0 e 10, che sono anche i momenti in cui rilascio job di T_1, non c'è nessun altro momento in cui c'è rilascio contemporaneo di job di T_1 e T_2.
T_3 avrà rilasci in 0, 3, 6, 9: in 0 ho istante critico, 6 e 9 sono critici ma il thm non li evidenzia.



Stesso esempio anche se i task non sono in fase: 6 è istante critico, è descritto dal teorema.
Quando c'è rilascio in fase, siccome priorità è fissa, la schedulazione prodotta risulta identica a qualsiasi schedulazione non in fase ⇒ mi interessa ricondurmi

a quando tutti i task sono in fase.

1.2 Schedulabilità per priorità fissa e tempi di risposta piccoli

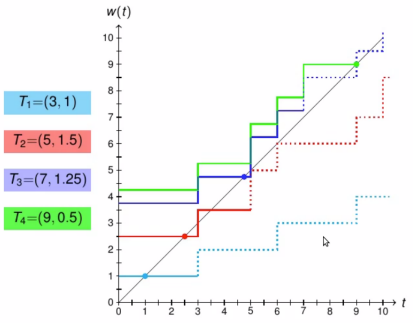
Supponiamo che in un sistema ho task a priorità fissa e tempi di risposta piccoli. Ordino i task per priorità decrescente, suppongo siano in fase all'istante t_0.
Ho i task T_1.....T_i e mi chiedo il tempo necessario per eseguire tutti i job dei task T_1.....T_i, nell'intervallo $[t_0, t_0+t]$ ($t \leq p_i$:
$w_i(t) = e_i + \sum_{k=0}^{i-1} \lceil \frac{t}{p_k} \rceil \cdot e_k$.
Somma si estende su tutti i task di priorità superiore di T_i, devo considerarli perché portano via tempo al job di T_i. Prendo k-esimo task: a t_0 tutti i task sono in fase, quindi rilascio sicuro un job, quando ne rilascio? Prendo il ceil di $\frac{t}{p_k}$,anche job rilasciato nel periodo dopo quello considerato mi ruba tempo; moltiplico tutto per e_k, il tempo che ci metto per completare i job.
Test di schedulabilità: dati job T_1.....T_i, in fase a t_0 con priorità decrescenti con T_1.....T_{i-1} effettivamente schedulabili. Il task T_i può essere schedulato nell'intervallo di tempo $[t_0, t_0+D]$ se $\exists$ $t \leq D_i$ tale che $w_i(t) \leq t$. IL mio scopo è sempre quello di verificare la schedulabilità del sistema, se ne trovo uno non schedulabile la mia analisi è finita, non ci faccio nulla col sistema di task.
Applicazione: ho T_1.....T_n con priorità decrescenti.
Considero un task alla volta: $\forall$ task T_i calcolo il valore della funzione di tempo necessario $w_i(t)$ per tutti i valori $t \leq D_i$ tali per cui t è un multiplo intero di p_k per k $\in$ {1,2....,i }.Funzione $w_i(t)$ sale a gradini, devo considerare valori per cui tale funzione cambia valori.
Se per almeno uno dei valori t vale che $w_i(t) \leq t$ allora T_i è effettivamente schedulabile. Altrimenti il test fallisce, ovvero n job di T_i potrebbe mancare la scadenza, ovvero la manca sicuro se c'è un rilascio di tutti i job in fase dei task di priorità superiore e tutti quei task hanno un tempo di esecuzione pari al loro worst-case.
Possono esserci casi fortuiti, quindi in ipotesi rilassate il test non conferma schedulabilità ma scheduler riesce, però il risultato non è rilevante.
Tanto vale fermarsi e riprogettare il sistema.
esempio: T_1=(3,1), T_2=(5,1.5), T_3=(7, 1.25), T_4=(9,0,5) e considero le funzioni di tempo necessario:

Esempio: T_1=(3, 1), T_2=(5, 1.5), T_3=(7, 1.25), T_4=(9, 0.5)

t	3	5	6	7	9
$w_1(t)$	1.0				
$w_2(t)$	2.5	3.5			
$w_3(t)$	3.75	4.75	6.25	7.25	
$w_4(t)$	4.25	5.25	6.75	7.75	9.0

Grafico per l'esempio precedente, ho la bisettrice del 1° quadrante, dire che $w_i(t)$ è $\leq$ t vuol dire che $w_i(t)$ sta sotto la bisettrice. La funzione è a scalini, non ha senso calcolarla, la applico nel periodo tra 0 e la fine del periodo.In T_2 la

funzione sale sopra la bisettrice, ma non è importante: devo verificare che sia sotto in un certo momento, se fosse sempre sopra non sarebbe schedulabile.
Ogni volta che c'è rilascio di un task a priorità superiore ⇒ ho gradino nella funzione di tempo necessario.



1.3 Massimo tempo di risposta

Massimo tempo di risposta W_i di T_i è il più piccolo valore prima della scadenza relativa t.c : t=w_i(t). Se l'equazione non ha soluzioni $\leq$ a D_i, allora qualche job di T_i mancherà la scadenza relativa.
Uso un algoritmo:

- $t^(1) = e_1$ in prima approssimazione
- Sostituisco nella funzione ed ottengo un nuovo valore $t^(k+1) = w_i(t^(k))$
- continuo ad iterare finché:
  - $t^(k+1) = t^(k)$ e $t^(k) \leq D_i \Rightarrow W_i = t^(k)$
  - $t^(k) \geq D_i$ e allora sono fuori scadenza

Ma dato che caso peggiore sono task in fase e dato che ho tutti i parametri sono noti, non sarebbe più facile provare a simulare la schedulazione? Sì, ma ci sono dei fattori che non ho considerato e che mi impediscono di simulare, esempio:

- Non è possibile determinare facilmente il worst case
- Il worst case cambia da task a task
- È difficile integrare nella simulazione altri fattori che possono essere considerati estendendo il test di schedulabilità.

In ogni caso, sia simulare il test che il test di schedulabilità stesso hanno la stessa complessità.

1.3.1 Task periodici con tempi di risposta arbitrari

Considero ora task con tempi di risposta arbitrari, che implica che:

- Un job non deve necessariamente prima che il job successivo dello stesso task sia eseguito
- è possibile che $D_i \geq$ di p_i
- Ci possono essere nello stesso istante più job di uno stesso task in attesa di essere eseguiti.
- Un job rilasciato contemporaneamente a tutti i job dei task con priorità maggiore non ha necessariamente il massimo t. di risposta possibile.

Assumo sempre che i job di uno stesso task hanno vincoli di precedenza impliciti fra di loro, ovvero sempre eseguiti FIFO.
Analizzo task per task: considero T_i (i precedenti sono schedulabili). Ho insieme task τ_i=T_1....T_i con priorità decrescente.
Definisco un intervallo totalmente occupato di un livello π_i un intervallo (t_0, t_1] tale che:

- all'istante t_0 tutti i job di τ_i rilasciatiti prima di t_0 sono stati completati
- All'istante t_0 un job di τ_i viene rilasciato.
- L'istante t_1 è il primo istante in cui tutti i job di τ_i rilasciati a partire da t_0 sono stati completati

È possibile che in un intervallo totalmente occupato il processore sia idle o esegua task non di τ_i? No: se fosse idle, l'intervallo terminerebbe prima, non può neanche eseguire task di priorità inferiore, quindi non può eseguire task al di fuori di τ_i
esempio: T_1, T_2, T_3.
Intervalli di T_3 non sono lunghi uguale, questo perché i rilasci di T_3 non sono in concomitanza con T_1 e T_2, posso dire che l'intervallo a lunghezza massimo quando i rilasci di tutti i task sono in fase.
Test di schedulabilità generale per tempi di risposta arbitrari è ancora basato sul caso peggiore, la differenza rispetto al test per tempi piccoli è che il primo job rilasciato contemporaneamente agli altri potrebbe non avereil massimo tempo di risposta.
Idea : $\forall$ T_i analizzo tutti i suoi job eseguiti nel primo intervallo totalmente occupato di livello π_i.
Come determino l'intervallo totalmente occupato:

- Inizio determinato dal rilascio dei primi job (in fase) dei task τ_i={T_1, ..., T_i}
- Lunghezza massima calcolata risolvendo iterativamente $t = \sum_{k=1}^{i} \lceil \frac{t}{p_k} \rceil \cdot e_k$. Molto simile alla funzione di tempo necessario, dico che aumento t fino a

che non trovo il valore dato dalla sommatoria, ovvero il primo t per cui il lavoro necessario per compiere tutti i task permette di eseguire tutti i task rilasciati nell'intervallo [t_0, t_0+t]

Quindi si procede nel seguente modo:

- Considero i task {T_1, ..., T_i} con priorità $\pi_1 < \pi_2 .... < \pi_i$, considero un task T_i alla volta cominciando da quello con la massima priorità, ovvero T_1
- Il caso peggiore per la schedulabilità di T_i: assumere che i task τ_i = {T_1, ..., T_i} sono in fase.
- Se il primo job di tutti i task in Tau_i termina entro il primo periodo del task ⇒ decidere se T_i è schedulabile si effettua controllando se $J_{i,1}$ termina entro la scadenza tramite la funzione di tempo richiesto $w_{i,1} := w_i(t)$
- Altrimenti almeno un primo job di Tau_i termina dopo il periodo del task, calcola la lunghezza t^L dell'intervallo totalmente occupato di livello π_i che inizia da t = 0.
- Calcolo i tempi di risposta massimi di tutti i job di T_i dentro l'intervallo totalmente occupato che sono $\lceil \frac{t^L}{p_i} \rceil$; il primo l'ho già calcolato.
- Decido se questi job sono schedulabili dentro l'intervallo totalmente occupato. Uso un lemma:
  Il tempo di risposta massimo $W_{i,j}$ del j-esimo job di T_i, in un intervallo totalmente occupato di livello π_i in fase è uguale al minimo t che soddisfa l'equazione t = $w_{i,j}$(t+(j-1)· p_i) - (j-1)· p_i, con $w_{i,j}(t) = j \cdot e_i + \sum_{k=1}^{i-1} \lceil \frac{t}{p_k} \rceil \cdot e_k$.
  Aggiungo un j che moltiplica e_i, devo verificare l'equazione nei punti multipli.

esercizio: T_1 = (ϕ_1,2,1,1), T_2 = (ϕ_2,3,1.25,4), T_3 = (ϕ_3,5,0.25,7)
Parto verificando T_1:
$w_1(t) = w_{1,1}(t) = e_1 = 1 = D_1$. Quindi è sicuramente schedulabile .
T_2:
$w_{2,1}(2) = e_1 + e_2 = 2.25 > 2$, quindi non va bene. Vado avanti:
$w_{2,1}(3) = 2 \cdot e_1 + e_2 = 3.25 > 3$. Non va ancora bene, proseguo:
$w_{2,1}(4) = 2 \cdot e_1 + e_2 = 3.25 \leq 4 \leq D_2$ quindi T_2 è schedulabile, ma ha completato oltre il periodo ⇒ non posso più considerare tempi piccoli, devo considerare gli intervalli totalmente occupati, uso l'equazione iterativa:
$t^{(}1) = e_1 + e_2 = 2.25$, sostituisco nella sommatoria,ed ottengo $t^{(}2) = 2 \cdot e_1 + e_2 = 3.25$, $t^{(}3) = 2 \cdot e_1 + 2 \cdot e_2 = 4.5$, $t^{(}4) = 3 \cdot e_1 + 2 \cdot e_2 = 5.5$, $t^{(}5) = 3 \cdot e_1 + 3 \cdot e_2 = 5.5 \Rightarrow t^{(}4) = t^L$, ovvero intervallo totalmente occupato di livello 2 è 5.5.
Ora calcolo quanti job di T_2 ci sono in (0, 5.5] = $\lceil \frac{t^L}{p_2} \rceil = 2$.
Veridico il secondo job di T_2:

$w_{2,2}(3) = 2{\cdot}e_1 + 2{\cdot}e_2 = 4.5 > 3$, no
$w_{2,2}(4) = 2{\cdot}e_1 + 2{\cdot}e_2 = 4.5 > 4$, ancora no.
$w_{2,2}(3) = 3{\cdot}e_1 + 2{\cdot}e_2 = 5.5 \leq 6 \leq p_2 + D_2 = 7$, quindi accetto il task.
Ora devo capire se posso accettare T_3, e considerare l'intervallo totalmente occupato di lvl 3:
$t^(1) = e_1 + e_2 + e_3 = 2.5$
$t^(2) = 2{\cdot}e_1 + e_2 + e_3 = 3.5$
$t^(3) = 2{\cdot}e_1 + 2{\cdot}e_2 + e_3 = 4.75$
$t^(4) = 3{\cdot}e_1 + 2{\cdot}e_2 + e_3 = 5.75$
$t^(5) = 3{\cdot}e_1 + 2{\cdot}e_2 + 2{\cdot}e_3 = 6$
$t^(6) = 3{\cdot}e_1 + 2{\cdot}e_2 + 2{\cdot}e_3 = 6 = t^L$
job di T_3 nell'intervallo (0,6]: $\lceil \frac{t^L}{p_3} \rceil = 2$. Considero i singoli job:
$w_{3,1}(2) = e_1 + e_2 + e_3 = 2.5 > 2$, no.
$w_{3,1}(3) = 2{\cdot}e_1 + e_2 + e_3 = 3.5 > 3$, no.
$w_{3,1}(4) = 2{\cdot}e_1 + 2{\cdot}e_2 + e_3 = 4.75 > 4$, no.
$w_{3,1}(5) = 3{\cdot}e_1 + 2{\cdot}e_2 + e_3 = 5.75 > 5$, no.
$w_{3,1}(6) = 3{\cdot}e_1 + 2{\cdot}e_2 + e_3 = 5.75 \leq 6 \leq D_3 = 7$. Posso accettare il job

$w_{3,2}(5) = 3{\cdot}e_1 + 2{\cdot}e_2 + 2{\cdot}e_3 = 6 > 5$, no.
$w_{3,2}(6) = 3{\cdot}e_1 + 2{\cdot}e_2 + 2{\cdot}e_3 = 6 \leq 6 \leq p_3 + D_3 = 12$ Accetto il job, e quindi il task.
Tutti i task sono schedulabili a prescindere dai loro task.

1.4 Condizioni di schedulabilità

Il test di schedulabilità generale determina se insieme di task è schedulabile o no, considerando worst case che è task in fase.
Ho dei limiti:

- Devo conoscere tutti i periodi, le scadenze ed i tempi d'esecuzione. Per validazione è necessario, ma no per implementazione di scheduler a priorità fissa. Se voglio aggiungere un task dovrei conoscere parametri che in fase di progettazione del sw non servono.
- Il risultato ottenuto non è valido se il task varia periodo, scadenza o tempo di esecuzione.
- È computazionalmente costoso, poco adatto per scheduling on-line.

Cerco di trovare delle condizioni di schedulabilità, confronto il test con la condizione, che è molto più semplice da calcolare e che può essere applicata anche se alcuni parametri non sono noti (esempio: condizione di EDF).
Mi chiedo se $\exists$ condizione di schedulabilità per algoritmi a priorità fissa:
Condizione di Liu-Layland: sistema τ di n task indipendenti ed interrompibili con scadenze relative uguali ai rispettivi periodi può essere effettivamente schedulato su un processore in accordo con RM se il suo fattore di utilizzazione

U_τ è $\leq$ a U_{RM}(n) = n·($2^{\frac{1}{n}}$-1)
Questo è il fattore di utilizzazione di RM, se considero: $\lim_{n\to\inf} U_{RM}(n) = \ln 2$, ovvero RM in generale garantisce di rispettare le scadenze pur di non caricare il processore per più del 69.3.
Ho un criterio per adottare RM negli scheduler real-time.
esempio:
T_1 = (1,0.25), T_2 = (1.25,0.1), T_3 = (1.5,0.3), T_4 = (1.75,0.07), T_5 = (2,0.1).
$U_\tau = 0.62 \leq 0.743 = U_{RM}(5) \Rightarrow$ è schedulabile con RM.
IL sistema T_1 = (3,1), T_2 = (5,1.5), T_3 = (7,1.25), T_4 = (9,0.5) ha fattore di utilizzazione $U_\tau = 0.867 > 0.757 = U_{RM}(4)$, forse non schedulabile.
È condizione sufficiente, difatti l'esempio 2 era quello precedente che è schedulabile se applico la funzione di tempo necessario.
L'alternativa a questo risultato è il test iperbolico: Un sistema τ di n task indipendenti ed interrompibili con scadenze relative uguali ai rispettivi periodi può essere effettivamente schedulato su un processore RM se $\prod_{k=1}^{n}(1+\frac{e_k}{p_k}) \leq 2$.
SI applica anche questo conoscendo solo fattore di utilizzazione dei task.
Correlazione con condizione di Liu-Layland: se gli n task hanno tutti lo stesso rapporto $\frac{e_k}{p_k}$ vuol dire che ciascun di questi usa una porzione uguale del processore.
Si può dimostrare che se questo è vero allora, assumendo $u_k = \frac{U_\tau}{n}$:
$\prod_{k=1}^{n}(1+\frac{e_k}{p_k}) \leq 2 \Leftrightarrow U_\tau$ *leq* n·($2^{\frac{1}{n}}$-1).
Se questo non è vero, esistono casi in cui il test iperbolico è soddisfatto, ma la condizione di Liu-Layland no; non esiste invece mai il viceversa.

1.5 Test per sottoinsiemi di task armonici

So che ,in generale RM è schedulabile se è soddisfatta condizione di Liu-Layland, ma so anche che su task armonici è ottimale. Suddivido insiemi di task in sottoinsiemi di task armonici fra loro.
Condizione di Kuo-Mok: se sistema τ di task periodici, indipendenti ed interrompibili con $p_i = D_i$ può essere partizionato in n_h sottoinsiemi disgiunti Z_1,....,Zn_h, ciascuno dei quali contiene task semplicemente periodici, allora il sistema è schedulabile con RM se:
$\sum_{k=1}^{n_h} U_{Z_k}(n_h)$ oppure se $\prod_{k=1}^{n_h}(1+U_{Z_k}) \leq 2$.
Se un sistema ha poche applicazioni molto complesse, è possibile migliorare la schedulabilità rendendo i task di ciascuna applicazione semplicemente periodici.
Esempio: 9 task con periodi 4,7 ,7 , 14, 16, 28, 32, 56, 64, fattore di utilizzazione di Liu-Layland è $U_{RM} = 0.720$
Considero i multipli di 2 e 7 e partizionando in due sottoinsiemi ottengo U_{Z_1} + $U_{Z_2} \leq U_{RM}(2) = 0.828$.

Il fattore di RM è in generale U_{RM}(n), ma posso farlo diventare pari ad 1

per task semplicemente periodici.
Miglioro U_{RM}(n) considerando quanto i periodi dei task sono vicini ad essere armonici:
$X_i = \log_2 p_i$ - $\lfloor \log_2 p_i \rfloor$ e $\zeta = \max_{1 \leq i \leq n} X_i$ - $\min_{1 \leq i \leq n} X_i$
Considero il valore frazionario del $\log_2$ e prendo tutti i task, di cui faccio differenza tra max e min di questi scarti decimali.
Teorema: nelle ipotesi della condizione di Liu-Layland, il fattore di utilizzazione di RM dipende dal numero di task n e da ζ è: $U_{RM}(n, \zeta) =$

- (n-1)$\cdot(2^{\frac{\zeta}{(n-1)}}$-1) + $2^{(1-\zeta)}$-1 se $\zeta < 1$ - $\frac{1}{n}$
- U_{RM}(n)

Quando si verifica il caso $\zeta = 0$? Quando $p_i = K \cdot 2^{x_i}$; non è vero il contrario
Variante: schedulabilità per scadenze arbitrarie. Se per qualche task la scadenza è più grande del periodo il limite è valido? Sì, però la formula è "pessimista": forse è possibile trovare valori di soglia superiori a U_{RM}.
Se invece per qualche task il periodo è più grande della scadenza non posso applicare Liu-Layland.
Teorema:
Un sistema τ di n task indipendenti, interrompibili e con scadenze $D_i = \delta p_i$ è schedulabile con RM se U_τ è $\leq$ a: $U_{RM}(n, \delta) =$

- δ(n-1)$\cdot(\frac{\delta+1}{\delta}^{\frac{1}{(n-1)}}$ - 1) per δ = 2,3,.....
- n($2\delta^{\frac{1}{n}}$-1) + 1 - δ per $0.5 \leq \delta \leq 1$
- δ per $0 \leq \delta \leq 0.5$

2 Schedulazione di job bloccanti e job aperiodici

Avevo un modello semplice, devo rilassare qualcuna delle ipotesi dovute al fatto che i job siano sempre sempre interrompibili, o che costo di context switching sia 0. Nella pratica molti fattori rallentano l'esecuzione di un job, che possono portare a mancata scadenza. Devo tenerne conto, divido in due genti classi:

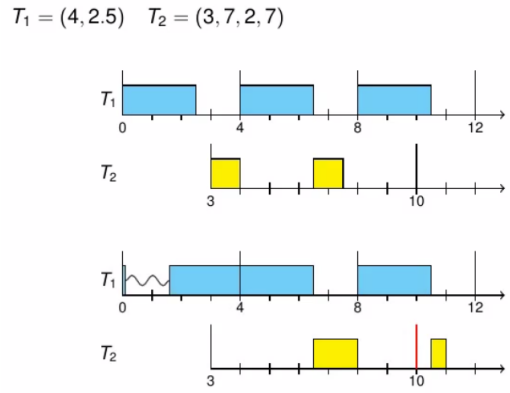
- Tempi di blocco: job non può essere eseguito nonostante il rilascio, per via di fattori esterni. Ad esempio: sul processore c'è un job non interrompibile, job rilasciato è quindi bloccato per un certo tempo. Modellati definendo b_i = tempo massimo di blocco, che tiene conto di tutti i tempi che fanno si che il job non può eseguire, va sottratto al tempo a disposizione del job.
- rallentamenti sistematici: ho calcolato il worst case di un job, ma a questo devo considerare il tempo che ci mette il job ad essere posto in esecuzione e ad essere tolto una volta completato, o anche il tempo che ci mette lo scheduler a decidere. Se questo tempo ha impatto pratico può avere senso modellarlo. Sommo al worst case del job.

2.1 Auto-sospensione

Un job rilasciato non può essere eseguito perché in attesa di eventi esterni, la cosa migliore da fare in questi casi è mettere in esecuzione un altro job. Si dice che il job si è auto-sospeso:

- job è un processo ed esegue operazione di accesso alla memoria di massa, ha senso sostituire il processo mentre questo attende i dati.
- attendo dati da rete/altri job
- attendo scadenza di un timer

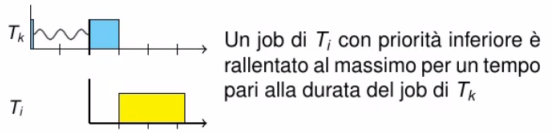
Nei SO questo tipo di operazioni sono chiamate operazioni bloccanti, nell'ambito real-time ci possono essere operazioni di auto-sospensione che però non è bloccante: in questo ambito ha senso attivo, ovvero un job ne blocca un altro. Anche in questo caso ci sono conseguenze su un altro job.
Supponiamo che ogni job di un task T_i si auto-sospende per un certo tempo x, in questo caso non appena rilasciato. Come schedulo: considero l'istante di rilascio come p_i-x, e la scadenza relativa come D_i-x.
Approccio semplificato, non funziona nel caso in cui i job si auto-sospendono solo all'inizio o per un tempo determinato, devo definire il tempo massimo di auto-sospensione b_i(ss).
esempio: T_1 = (4, 2.5) T_2 = (3.7, 2, 7) schedulato con RM. Se primo job di T_1 si auto-sospende subito dopo il rilascio, le cose possono andare male: il primo job del task T_2 manca la scadenza, job di T_1 si risveglia in modo che per completare occupa tutto il suo periodo, quindi quando job di T_2 comincia esecuzione di porta avanti ma non riesce a finire.



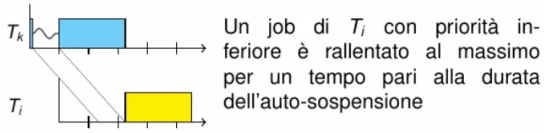
Ho impatto sui job di priorità inferiore: anche se job si auto-sospende on no lo fa, se c'è job di priorità superiore non è danneggiato, ma quelli di priorità inferiore si.

2.1.1 Rallentamento dovuto all'auto-sospensione

1° caso: il tempo di auto-sospensione di un job è maggiore della durata del job: job di T_i con priorità inferiore è rallentato al massimo per un tempo pari alla durata del job di T_k È il worst case: job T_i non riesce ad arrivare mente job di T_k è in auto-sospensione



2° caso: il tempo di auto-sospensione di un job è minore della durata del job. Un job di T_i con priorità minore è rallentato al massimo per un tempo pari alla durata dell'auto sospensione.



2.1.2 Tempo massimo di sospensione di blocco per auto-sospensione

Dato task T_k chiamo x_k il tempo massimo di sospensione di ciascun job di T_k, questo è un parametro del sistema.
Prendo task T_i con priorità minore, il rallentamento inflitto ad un job T_i da un job di T_k è minore o uguale ad x_k e minore o uguale ad e_k: $b_i(ss) = x_i + \sum_{k=1}^{i-1} min(e_k, x_k)$.
Manca qualcosa, sto assumendo che un job si auto-sospenda una volta sola, ma non c'è nessun motivo reale per cui questo sia vero: job può auto-sospendersi più volte, devo contare il numero di volte. Devo definire anche il massimo numero di volte k_i in cui un job di T_i si sospende.
Difatti:

- si può verificare un blocco da parte di un processo non interrompibile
- si ha un rallentamento dovuto allo scheduler ed al costo del context switching

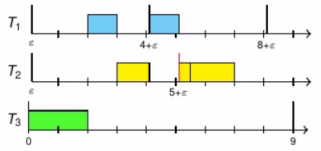
2.2 Non interrompibilità dei job

Assunzione irrealistica che i job non siano interrompibili, esistono sempre istanti in cui il job non è interrompibile:

- se sta operando su area di memoria critica
- se sta interagendo con dispositivo hardware

- job esegue syscall, e ci sono chiamate di sistema che non possono essere interrotte.Job diventa non interrompibile fino alla conclusione del SO.
- costo del context switch è troppo elevato

Un job J_i è bloccato per non interrompibilità quando è pronto per essere eseguito, ma non può perché è in esecuzione un job non interrompibile.
Quando si verifica questo fenomeno, si parla di inversione di priorità quando la priorità del job in esecuzione è minore di quella del job pronto per l'esecuzione. esempio: $T_1 = (\epsilon, 4, 1, 4)$, $T_2 = (\epsilon, 5, 1.5, 5)$, $T_3 = (9, 2)$. Qualunque sia l'algoritmo, all'istante 0 viene messo in esecuzione job di T_3, inoltre $U = 0.77$ ed è schedulabile per EDF e RM, ma solo se job sono non interrompibili. Suppongo che T_3 non sia interrompibile, conclude nell'istante 2, quindi tra $[\epsilon,2]$ blocca due job con priorità maggiore. Nell'intervallo tra $[2, 5+\epsilon]$ eseguo 3 job, 2 di T_1 ed uno di T_2, ma non c'è abbastanza tempo e quindi T_2 manca la scadenza.



Come faccio a modellare che il job è non interrompibile, devo capire la durata massima di non interrombilità di un job: sia
$Theta_k$ il tempo di esecuzione massimo della più lunga sezione non interrompibile dei job di T_k. Sia $b_i(np)$ il tempo massimo di blocco per non interrompibilità, che è tempo subito da un job a causa dei job di priorità inferiore, quando vale? $b_i(np) = \max\{\Theta_k$: per ogni task T_k di priorità minore a T_i $\}$: suppongo che c'è job di alta priorità rilasciato, ho sul processore job di priorità inferiore T_k appena entrato nella sezione critica non interrompibile più lunga, subisco rallentamento di Θ_k, ma appena finisce la sezione lo scheduler da la priorità a me, non importa quanto solo lunghe le sezioni degli altri job: caso peggiore è che vengo rilasciato quando il job che ha il Θ_k più lungo entra in esecuzione.
Il tempo massimo di blocco totale dipende da entrambe i due tempi di blocco: $b_i = b_i(ss) + (K_i+1) \cdot b_i(np)$. Considero numero massimo di volte per cui il job J_i si sospende, il $+1$ è il fatto che la prima volta deve essere rilasciato sia che si auto-sospende che non.

2.3 Cambi di contesto

Come modellare rallentamenti dovuti al context switch: CS= context switch time, per ora ci metto anche tempo necessario per lo scheduler per prendere decisione.
Allungo il worst case del job: calcolato quando non c'è nulla che interferisce col job. worst case è $e'_i = e_i + 2 \cdot (K_i + 1) \cdot CS$. job deve subire almeno due cambi di contesto: quando viene messo in esecuzione e quando viene tolto

dall'esecuzione.Ma ogni volta che il job si auto-sospende c'è un altro cambio di contesto: per essere tolto e poi per essere rimesso; K_i = n° volte che il job si auto-sospende.
Alle volte non è utile modellare il context switch, però in altri casi è essenziale farlo: LST si basa sullo slack rimanente, quindi ci sono molti cambi di contesto e l'overhead è significativo ed è doveroso modellarli. Con LST è anche spesso difficile capire qual'è numero massimo di context switch di job, ma ci sono algoritmi come EDF altrettanto buoni, in un sistema real-time parametro cruciale: i job devono rispettare le scadenze; utiele vederlo in teoria ma non in pratica.

2.4 Test di schedulabilità per job bloccanti

Come faccio ad usare i parametri definiti nel processo di validazione: o uso test di schedulabilità o uso condizioni di schedulabilità.
Idea è che tempo disponibile per completare per ciascun job va diminuito del tempo massimo per cui quel job può rimanere bloccato, definisco tempo di blocco come tempo max aggiuntivo. La funzione di tempo massimo richiesto diventa:
$w_i(t) = e_i + b_i + \sum_{k=1}^{i-1} \lceil \frac{t}{p_k} \cdot e_k \rceil$ per $0 < t \leq \min(D_i, p_i)$. Ho meno tempo a disposizione per completare il job, sommo b_i.
Stesso si applica al test di schedulabilità generale:
$w_{i,j}(t) = j \cdot e_i + b_i + \sum_{k=1}^{i-1} \lceil \frac{t}{p_k} \cdot e_k$ per $(j-1) \cdot p_i < t \leq w_{i,j}(t)$. non devo moltiplicare b_i per j: il 3° job di T_i è sempre $3 \cdot e_i$, ma sto cercando di capire quanto tempo rimane al 3° job, perché questo viene bloccato solo per b_i. Il blocco è qualcosa che considero soltanto quando devo studiare la schedualbilità del singolo job ed è relativa solo al singolo job. Non ha senso considerarla per tutti i task insieme, si fa sempre studio task per task.

2.5 Condizioni di schedualbilità per task bloccanti +a priorità fissa

Sia dato sistema di n task T ed un algoritmo a priorità fissa X, con fattore di utilizzazione $U_X(n)$. Sappiamo che il sistema è effettivamente schedulabile se $U_T \leq U_X(n)$, a condizione che i task non blocchino mai. Come adatto la condizione per task a priorità fissa ma che bocchino? Non posso più usare solo le condizioni di schedualbilità, perché ciascun job può bloccare con misura differente, quindi devo farlo per un task alla volta. Nel caso peggiore, ogni job di T_i impiega un tempo $e_i + b_i$ per completare l'esecuzione. Posso modellare questo tempo come tempo di esecuzione in più che il job deve subire: dato un task T_i, calcolo utilizzazione totale fino alla priorità i, dato task calcolo utilizzazione totale fino alal priorità i:
$\sum_{k=1}^{i} \frac{e_k}{p_k} + \frac{b_i}{p_i} \leq U_X(i)$. Task di priorità inferiore non possono incidere sulla priorità del task, o meglio lo faranno solo se sono non bloccanti ma lo sto già

considerando. Guardo solo ai task con priorità maggiore, considero come n° task solo fino ad i, considero solo $U_X(i)$, man mano arriverò ad $U_X(n)$. Applico anche ad EDF, considero task per task, parlo in generale di densità ed uso approccio simile al precedente: task per task questo è schedualbile se:
$\sum_{k=1}^{n} \frac{e_k}{min(D_k,p_k)} + \frac{b_i}{min(D_i,p_i)} = \Delta_\tau + \frac{b_i}{min(D_i,p_i)} \leq 1.$
Non sto parlando di task a priorità fissa, ogni job del sistema può avere priorità che precede il job in questione: di fatto, non posso applicare sommatoria solo a task a priorità superiore ma devo applicare a tutti i task del sistema, quindi arrivare alla densità del sistema. Alla densità contribuisce anche il task in questione che è $\frac{e_i}{min(D_i,p_i)}$, a cui aggiungo anche $\frac{b_i}{min(D_i,p_i)}$ poiché è come se il tempo di esecuzione del job del task è aumentato di b_i.
Problema è definire i tempi massimi di blocco se i task non hanno priorità fissata, la priorità è del job.
Teorema (Baker, 1991): in una schedualzione EDF un job con scadenza relativa D può bloccare un altro job con scadenza relativa D' solo se D > D'.
Dim: se il job con scadenza relativa D blocca quello con D', vuol dire che la sua priorità è inferiore: bloccare ha il senso che un job a priorità inferiore sta togliendo tempo ad uno a priorità superiore, quindi d > d' (scadenze assolute), per poter bloccare il processore deve averlo messo in esecuzione prima e quindi r < r' ⇒ D = d-r > d'-r' = D'. Ho una soluzione: posso ordinare i task per scadenze relative crescenti, ed applico la formula di b_i per i task con priorità fissa.
Caso dell'auto-sospensione è difficile, quindi come realizzare il teorema di Baker? Thm non è più valido: se per esempio job J' ha priorità più alta di un job J. Se J' comincia ad eseguire e si auto-sospende: prima di tornare in esecuzione comincia job di priorità più bassa. L'ipotesi che r sia < r' non è più vera, può essere dopo r' semplicemente perché il job si è autosospeso: dovrei applicare al tempo r'+tempo dopo la sospensione.
Posso applicare il ragionamento a r'+x'+e': di quanto tempo r può precedere r', sicuramente di x'+e'.Può non precedere r', ma r'+x'+e' è la massima distanza che posso avere fra r ed r'. Formulo teorema di Baker con auto-sospensione: in una schedulazione EDF, un job con scadenza relativa D può bloccare un altro job con scadenza relativa D' e tempo massimo di esecuzione x' solo se D > D'-x'-e'.
Dato che entrambi i job possono auto-sospendersi, è possibile che i due task possano bloccarsi a vicenda non ho più ordinamento totale.

2.6 Schedulazione basata su tick

Fin'ora ho visto scheduler event-driven: viene eseguito quando si verifica un evento rilevante. In pratica, è più semplice realizzare uno scheduler time-driven, ovvero che si attiva ad interruzioni periodiche: svantaggio è che tutti i tempi nel sistema avranno granularità pari alla dimensione del mio tick.
Il riconoscimento di un evento come il rilascio di un job può essere differito fino al tick successivo, è come se ci fosse inversione di priorità. Definisco job

pendenti, ovvero che sono stati rilasciati ma che lo scheduler non ha ancora preso in considerazione perché non è scattato il tick, e quelli eseguibili, ovvero quelli piazzati dallo scheduler, ho due code per le rispettivi due classi. Scheduler sposta job da coda dei job pendenti a coda dei job eseguibili.Quando jop termina, so già qual'è il prossimo da eseguire: sarà quello successivo nella coda dei job eseguibili. Se arriva job, questo viene messo nella coda dei job pendenti.

2.6.1 Test schedulabilità per priorità fissa con tick

Come posso applicare il test di schedulabilità ad uno scheduler a priorità fissa basato su tick?
Considero scheduler che si attiva con periodicità p_0, esegue in tempo e_0 il controllo della coda di job pendenti e con CS_0 trasforma un job da pendente a eseguibile.
Per controllare la schedulabilità di T_i

- Devo aggiungere task per controllare schedulabilità di un task $T_0 = (p_0, e_0)$ a priorità massima.
- Devo modellare il fatto che qundo arriva job, questo va prima o poi trasformato da pendente ad eseguibile.
  - per tutti i task a priorità inferiore rispetto ai job di T_i, per cui devo tenere conto del fatto che lo scheduler interverrà e trasformerà il job pendente in un job nella coda eseguibile. Oltre ai job di priorità inferiore, che vanno da i+1 a n, aggiungo un numero corrispondete di task $T_{0,k} = (p_k, CS_0)$ per ogni k = i+1,..,n, con priorità maggiore di T_1, ma che hanno periodicità CS_0, ovvero il tempo che ci mette il processore a trasformare i job in eseguibile da pendenti.
  - job a priorità superiore, aggiungo a tutti i task di priorità superiore o uguale ad $(K_k + 1) \cdot CS_0$, considero K_k perché ogni volta che mi risveglio devo essere spostato da pendente ad eseguibile. Aggiungo questi valori ad e_k per ogni k = 1,2...,i.

  Perché non considero le auto-sospensione per i task a priorità inferiore? Perché task inferiore non viene mai eseguito al posto mio, pago solo il primo rilascio, perché fin quando io sono eseguibile, quelli con meno priorità di me non hanno possibilità di essere eseguiti prima di me.
- Devo anche considerare il tempo di blocco per non-interrompibilità, anche se tutti i miei job sono sempre non interrompibili. $b_i(np) = (\lceil max_{i+1 \leq k \leq n} \frac{\Theta_k}{p_0} \rceil + 1) \cdot p_0$. max Θ_k moltiplicato il p_0 diventa il max di tutti i Θ_k di priorità inferiore, in più c'è un p_0. esempio:
  ho lo shceduler che viene invocato con periodo p_0. Ad un certo istante viene rilasciato il job del task T_i, mi metto nel worst case, ovvero il job T_i arriva un infinitesimo dopo che lo scheduler ha finito di controllare la coda dei job pendenti, quindi fino al prossimo p_0 non potrò eseguire il job.

In questo periodo, prima che possa intervenire lo shceduler, si continua ad eseguire un job di priorità inferiore di T_k e questo job entra nella regione interrompibile all'interno del periodo p_0 in cui è arrivato T_i, task T_k continua l'esecuzione per un numero di periodi pari a $\frac{\Theta_k}{p_0}$. Solo quando scheduler interviene si rende conto che job di T_k è diventato interrompibile e può entrare T_1, e c'è la parte intera superiore perché se il n° di periodi non è intero,anche la frazione non completata porta via tempo e devo aspettare comunque il periodo successivo; il +1 è il rilascio, almeno un periodo lo devo aspettare anche se non ho sezione critica, il job è arrivato prima.

esempio: $T_1 = (0.1, 4, 1, 4.5)$, $T_2 = (0.1, 5, 1.8, 7.5)$, $T_3 = (0, 20, 5, 19.5)$ non interrompibile in $[r_3, r_3+1.1]$. Scheduler ha $p_0 = 1$, $e_0 = 0.05$, $CS_0 = 0.06$.
Faccio analisi dei singoli task:
Verifico T_1: sistema equivalente è $T_0 = 1{,}0.05)$, $T_{0,2} = (5{,}0.06)$, $T_{0,3} = (20{,}0.06)$, $T_1 = (4{,}1.06)$, $b_1 = 3$.
$w_1(t) = 1.06 + \lceil \frac{t}{1} \rceil 0.05 + \lceil \frac{t}{5} \rceil 0.06 + \lceil \frac{t}{20} \rceil 0.06$.
$w_1(4.06) = 4.43 \leq w_1(4.43) \leq 4.5$, quindi ok.
Procedo per T_2 e T_3 sempre considerando il sistema equivalente.
$w_2(4.86) = 7.29 \leq w_2(7.29) \leq 7.5$, quindi ok.
$w_3(6.06) = 12.25$, $w_3(12.25) = 16.53$, $w_3(16.53) = 19.65$, $w_1(19.65) = 19.8 > 19.5$, quindi no.
Devo concludere che il sistema non è validato, e va riprogettato.

2.6.2 Condizione di schedulabilità su tick

Per ciascun T_i da controllare faccio quanto segue:

- Uso il thm Baker ed ordino per scadenze relative crescenti
- aggiungo un task $T_0 = (p_0, e_0)$ di massima priorità
- Aggiungo $(K_k+1) \cdot CS_0$ al tempo i esecuzione e_k, devo farlo per tutti i task: i blocchi hanno una certa relazione ma non ho priorità fissate, quindi ogni job può portare via tempo ad un altro job nel sistema
- Tempo di blocco è $b_i(np) = (\lceil max_{i+1 \leq k \leq n} \frac{\Theta_k}{p_0} \rceil + 1) \cdot p_0$.

Nell'esempio di prima, ottengo densità totale $\Delta \simeq 0.95$., verifico T_1: prendo Δ e sommo $\frac{3}{4}$, ovvero tempo di blocco diviso periodo di T_1. Ottengo $1.69 >$, quindi non schedualbile. Mi posso fermare: basta trovare un task non schedulabile.

2.7 Schedulazione priority-driven di job aperiodici

Mi pongo il problema di dover gestire job che arrivano ad istanti di tempo non predicibili:

- Job aperiodici sfot-rt: non faccio nulla, voglio però che completino nel miro tempo possibile.

- Job aperiodici hard-rt: tempi di arrivo sconosciuti, durata sconosciuta e scadenze hard.

Se non ho nessuna ipotesi su tempi di arrivo ed esecuzioni non posso prendere impegni: potrà sempre arrivare qualcosa che non mi permette di rispettare le scadenze.
Richiedono algoritmi differenti, però devono essere corretti ed ottimali: le scadenze vanno rispettate, i job aperiodici hard-rt va rifiutato se non è possibile garantirne le scadenze.Inoltre: tempi di risposta dei job soft-rt non hanno scadenze ma vanno minimizzato i tempi di risposta.

2.7.1 Schedulazione di job aperiodici soft RT in background

Metto in coda apposta e quando job è in background eseguo il job aperiodico in cima alla coda. Algoritmo è corretto, i task periodici non sono influenzati, ma non è ottimale: ritardo job aperiodici senza motivo.esempio:
$T_1 = (3,1)$, $T_2 = (10,4)$ job aperiodico A con rilascio 0.1 e durata 0.8.

2.7.2 Schedulazione di job aperiodici soft RT interrupt-driven

Esegui job aperiodici nel mo,mento in cui li rilasci, algortimo non è corretto ma è ottimo: job aperiodici finiscono nei tempi minimi, ma task periodici possono mancare le scadenze.

2.7.3 Schedualzione di job aperiodici soft RT con slack stealing

Algoritmo esegue i job aperiodici in anticipo rispetto a quelli periodici finché c'è uno slack globale positivo.
È corretto perché i job periodici non perdono le scadenze. È ottimale, solo per job aperiodico in cima alla coda. Svantaggio è che tenere uno slack globale in uno scheduler priority divern è difficile. Job aperiodico riprende quando lo slack torna positivo.

2.7.4 Schedulazione di job aperiodici soft RT con polling

Algoritmo basato su polling, ovvero nel sys di task periodici introduco server di polling, a cui do un certo periodo p_s e tempo di esecuzione e_s e gli do priorità massima,così da ridurre i tempi di risposta dei job aperiodici. Server controlla coda job aperiodici, se vuota si auto-sospende fino a prossimo tick, altrimenti esegue per al più e_s unità di tempo, per poi auto-sospendersi.
È corretto se dimensiono il poller in modo tale che il suo fattore di utilizzazione non ecceda quello dell'algoritmo di schedulazione;come aggiungere un task periodico.
Non è ottimale, job aperiodico può arrivare subito dopo esecuzione del poller. Nell'esempio forse potevo anticipare l'esecuzione del job A senza far mancare le scadenze. Posso migliorare le capacità del server? Sì.