Sistemi Embedded e Real-time (M. Cesati)

Compito scritto del 22 settembre 2009

Esercizio 1. Il seguente sistema di task periodici è schedulato su un processore con un algoritmo "cyclic schedule": $T_1 = (4, 1, 8)$, $T_2 = (3, 1)$, $T_3 = (6, 5, 1, 5)$. (Si noti che il task T_3 ha fase 6, mentre gli altri task hanno fase 0.)

(a) Determinare una dimensione intera appropriata per il frame assumendo che i job non sono interrompibili.

(b) Determinare una schedulazione ciclica valida dei task periodici nell'intervallo $[0, 48]$ utilizzando la dimensione del frame determinata al punto (a), ovvero dimostrare formalmente che una tale schedulazione ciclica valida non esiste.

Esercizio 2. Si consideri un sistema di task periodici e indipendenti schedulati su un singolo processore: $T_1 = (4, 1, 3)$, $T_2 = (6, 2)$, $T_3 = (8, 3, 10)$. I job del task T_1 sono sempre interrompibili; i job del task T_2 sono interrompibili nel 90% del proprio tempo di esecuzione; i job del task T_3 sono interrompibili nel 95% del proprio tempo di esecuzione.

(a) Assumendo che nessun job si auto-sospende, dimostrare analiticamente che il sistema è schedulabile con RM.

(b) Dimostrare analiticamente che il sistema è ancora schedulabile con RM qualora si assuma che ciascun job di T_2 può auto-sospendersi per una sola volta e rimanere sospeso al massimo per 1.7 unità di tempo.

(c) Con riferimento al punto (b), spiegare il motivo per cui il task T_2 non è più necessariamente schedulabile se ciascun job di T_2, pur mantenendo il tempo massimo totale di sospensione uguale a 1.7 unità di tempo, può auto-sospendersi per due volte (invece di una).

Esercizio 3. Un sistema deve eseguire cinque job $J_1, \ldots, J_5$, con J_i di priorità maggiore di J_k se $i < k$. I job utilizzano tre risorse condivise R_1, R_2 e R_3, e le relative sezioni critiche sono: $J_1 : [R_1; 2]$, $J_2 : [R_1, 4]$, $J_3 : [R_2; 2]$, $J_4 :$ nessuna, $J_5 : [R_2; 3\,[R_3; 2]\,]$.

(a) Quali sono i tempi massimi di blocco dei job per conflitto di risorse utilizzando il procotollo NPCS?

(b) Quali sono i tempi massimi di blocco dei job per conflitto di risorse utilizzando il procotollo priority-inheritance?

(c) Il tempo di blocco $b_2(\mathrm{rc})$ di J_2 diminuisce nel protocollo priority inheritance rispetto al protocollo NPCS; spiegarne il motivo.

Sistemi Embedded e Real-time (M. Cesati)

Soluzioni del compito scritto del 22 settembre 2009

Esercizio 1. *Il seguente sistema di task periodici è schedulato su un processore con un algoritmo "cyclic schedule":* $T_1 = (4, 1, 8)$, $T_2 = (3, 1)$, $T_3 = (6, 5, 1, 5)$. *(Si noti che il task* T_3 *ha fase 6, mentre gli altri task hanno fase 0.)*

(a) *Determinare una dimensione intera appropriata per il frame assumendo che i job non sono interrompibili.*

- Vincolo sulle fasi dei task: per definizione, il primo job di ciascun task deve essere rilasciato esattamente all'inizio di un frame. In altri termini, la fase di ciascun task deve essere un multiplo intero non negativo della dimensione del frame.

  Poiché T_3 ha fase 6, sei deve essere un multiplo intero di f. Gli altri task con fase 0 non pongono alcun vincolo.

- Vincolo sui tempi d'esecuzione dei job:

$$f \geq \max\{1, 1, 1\} = 1$$

- Vincolo sulla divisibilità della lunghezza dell'iperperiodo ($H = \text{mcm}\{3, 4, 5\} = 60$):

$$f \in \{1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60\}.$$

  Considerando i vincoli precedenti: $f \in \{1, 2, 3, 6\}$.

- Vincolo sul task T_1 ($2f - \gcd\{4, f\} \leq 8$):

$$\begin{aligned} 2 \cdot 1 - \gcd\{4, 1\} &= 1 \leq 8 \\ 2 \cdot 2 - \gcd\{4, 2\} &= 2 \leq 8 \\ 2 \cdot 3 - \gcd\{4, 3\} &= 5 \leq 8 \\ 2 \cdot 6 - \gcd\{4, 6\} &= 10 > 8 \qquad \Rightarrow f \neq 6 \end{aligned}$$

- Vincolo sul task T_2 ($2f - \gcd\{3, f\} \leq 3$):

$$\begin{aligned} 2 \cdot 1 - \gcd\{3, 1\} &= 1 \leq 3 \\ 2 \cdot 2 - \gcd\{3, 2\} &= 3 \leq 3 \\ 2 \cdot 3 - \gcd\{3, 3\} &= 3 \leq 3 \end{aligned}$$

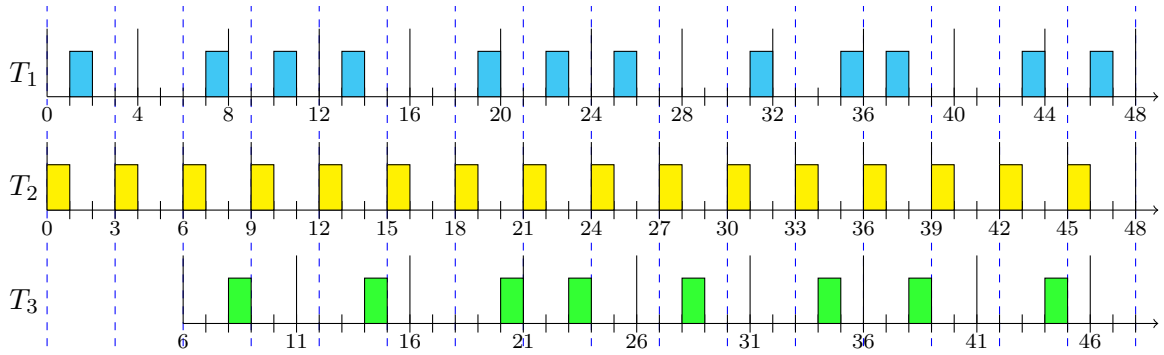
- Vincolo sul task T_3 ($2f - \gcd\{5, f\} \leq 5$):

$$2 \cdot 1 - \gcd\{5, 1\} = 1 \leq 5$$
$$2 \cdot 2 - \gcd\{5, 2\} = 3 \leq 5$$
$$2 \cdot 3 - \gcd\{5, 3\} = 5 \leq 5$$

Qualunque valore in $\{1, 2, 3\}$ è accettabile come dimensione del frame. Per ridurre l'overhead dovuto all'esecuzione dello scheduler è preferibile adottare come dimensione del frame $f = 3$.

(b) *Determinare una schedulazione ciclica valida dei task periodici nell'intervallo* $[0, 48]$ *utilizzando la dimensione del frame determinata al punto (a), ovvero dimostrare formalmente che una tale schedulazione ciclica valida non esiste.*

Un esempio di schedulazione ciclica valida è il seguente:



Equivalentemente, la schedulazione è descrivibile elencando i blocchi di schedulazione:

$L(0) = T_2, T_1$	$L(1) = T_2$	$L(2) = T_2, T_1, T_3$	$L(3) = T_2, T_1$
$L(4) = T_2, T_1, T_3$	$L(5) = T_2$	$L(6) = T_2, T_1, T_3$	$L(7) = T_2, T_1, T_3$
$L(8) = T_2, T_1$	$L(9) = T_2, T_3$	$L(10) = T_2, T_1$	$L(11) = T_2, T_3, T_1$
$L(12) = T_2, T_1, T_3$	$L(13) = T_2$	$L(14) = T_2, T_1, T_3$	$L(15) = T_2, T_1$

Esercizio 2*. Si consideri un sistema di task periodici e indipendenti schedulati su un singolo processore:* $T_1 = (4, 1, 3)$, $T_2 = (6, 2)$, $T_3 = (8, 3, 10)$. *I job del task* T_1 *sono sempre interrompibili; i job del task* T_2 *sono interrompibili nel 90% del proprio tempo di esecuzione; i job del task* T_3 *sono interrompibili nel 95% del proprio tempo di esecuzione.*

(a) *Assumendo che nessun job si auto-sospende, dimostrare analiticamente che il sistema è schedulabile con RM.*

Poiché le scadenze relative di T_1 e T_3 non coincidono con i rispettivi periodi non è possibile utilizzare le condizioni di schedulabilità.

Applichiamo il test di schedulabilità calcolando la funzione di tempo necessario

$$w_i(t) = e_i + b_i + \sum_k \left\lceil \frac{t}{p_k} \right\rceil \cdot e_k$$

ove la sommatoria è estesa a tutti i task con priorità maggiore di quella del task in esame.

Il tempo di blocco totale b_i è dovuto esclusivamente alla non-interrompibilità dei job; assumendo che i job non si auto-sospendono si ha:

$$b_i = b_i(np) = \max\{\, \theta_k \,:\, T_k \text{ di priorità inferiore a } T_i \,\}\,.$$

Nel caso peggiore, la durata della più lunga sezione di codice non interrompibile θ_k per il task T_k è data da:

$$\theta_1 = 0, \qquad \theta_2 = (1 - 0.90) \cdot 2 = 0.2, \qquad \theta_3 = (1 - 0.95) \cdot 3 = 0.15\,.$$

Di conseguenza:

$$b_1 = 0.2, \qquad b_2 = 0.15, \qquad b_3 = 0\,.$$

- Test di schedulabilità per T_1: $w_1(t) = 1 + 0.2 = 1.2 \leq 3$ $\Rightarrow$ schedulabile.
- Test di schedulabilità per T_2: $w_2(t) = 2 + 0.15 + \lceil t/4 \rceil$;
  $w_2(4) = 2.15 + 1 = 3.15 \leq 4$ $\Rightarrow$ schedulabile.
- Test di schedulabilità per T_3: $w_3(t) = 3 + \lceil t/4 \rceil + \lceil t/6 \rceil \cdot 2$;
  $w_3(4) = 6 > 4$
  $w_3(6) = 7 > 6$
  $w_3(8) = 9 > 8$
  $w_3(10) = 10 \leq 10$ $\Rightarrow$ schedulabile.
- Poiché il primo job di T_3 termina dopo il rilascio del successivo job di T_3, è necessario controllare anche la schedulabilità di tutti gli altri job di T_3 in un intervallo totalmente occupato.
- La lunghezza dell'intervallo totalmente occupato si ottiene risolvendo iterativamente $t = \lceil t/4 \rceil + \lceil t/6 \rceil \cdot 2 + \lceil t/8 \rceil \cdot 3$; il valore iniziale è la somma dei tempi d'esecuzione: $t_0 = 6$; $t_1 = 2 + 2 + 3 = 7$; $t_2 = 2 + 4 + 3 = 9$; $t_3 = 3 + 4 + 6 = 13$; $t_4 = 4 + 6 + 6 = 16$; $t_5 = 4 + 6 + 6 = 16 = t_4$.

- Numero di job di T_3 nell'intervallo totalmente occupato: $\lceil 16/8 \rceil = 2$.
- Test per il secondo job di T_3 utilizzando $w_{3,2}(t) = 2 \cdot 3 + \lceil t/4 \rceil + \lceil t/6 \rceil \cdot 2$, per $t \in [8, 10 + 8]$:

  $w_{3,2}(8) = 12 > 8$
  $w_{3,2}(10) = 13 > 10$
  $w_{3,2}(12) = 13 > 12$
  $w_{3,2}(16) = 16 \leq 16 \qquad \Rightarrow$ schedulabile.

Il sistema di task è perciò schedulabile con RM.

(b) *Dimostrare analiticamente che il sistema è ancora schedulabile con RM qualora si assuma che ciascun job di T_2 può auto-sospendersi per una sola volta e rimanere sospeso al massimo per* 1.7 *unità di tempo.*

Poiché

$$b_i(\text{ss}) = x_i + \sum_{k=1}^{i-1} \min(e_k, x_k),$$

si ha: $b_1(\text{ss}) = 0, \quad b_2(\text{ss}) = 1.7 + \min(1, 0) = 1.7, \quad b_3(\text{ss}) = 0 + \min(2, 1.7) = 1.7\,.$

- I tempi massimi di blocco dei job di T_1 non variano, perciò l'analisi condotta al punto (a) relativa alla schedulabilità del task T_1 resta valida.
- Il tempo massimo di blocco di un job di T_2 è dato da

  $$b_2 = b_2(ss) + (K_2 + 1) \cdot b_2(np)\,;$$

  poiché $b_2(ss) = 1.7$, $K_2 = 1$ e $b_2(np) = 0.15$ si ottiene $b_2 = 2$.
- Test di schedulabilità per T_2: $w_2(t) = 2 + 2 + \lceil t/4 \rceil$;

  $w_2(4) = 4 + 1 = 5 > 4$
  $w_2(6) = 4 + 2 = 6 \leq 6 \qquad \Rightarrow$ schedulabile.
- Analogamente, dato che $b_3(\text{np}) = 0$, il tempo massimo di blocco di un job di T_3 è dato da $b_3 = b_3(ss) = 1.7$.
- Test di schedulabilità per il primo job di T_3: $w_3(t) = 3 + 1.7 + \lceil t/4 \rceil + \lceil t/6 \rceil \cdot 2$;

  $w_3(4) = 4.7 + 1 + 2 = 7.7 > 4$
  $w_3(8) = 4.7 + 2 + 4 = 10.7 > 8$
  $w_3(10) = 4.7 + 3 + 4 = 11.7 > 10$

  Se ne deduce che il task T_3 non è più necessariamente schedulabile.

(c) *Con riferimento al punto (b), spiegare il motivo per cui il task T_2 non è più necessariamente schedulabile se ciascun job di T_2, pur mantenendo il tempo massimo totale di sospensione uguale a 1.7 unità di tempo, può auto-sospendersi per due volte (invece di una).*

Ogni volta che un job si auto-sospende può essere bloccato in fase di riavvio a causa della non interrompibilità dei job di priorità inferiore. In pratica, per ciascuna auto-sospensione il job può essere ulteriormente ritardato per un tempo pari a $b_i(np)$. Nel caso del task T_2, il tempo massimo di blocco sale a $b_2 = 2 + 0.15 = 2.15$, e di conseguenza la funzione di tempo necessario diventa $w_2(t) = 4.15 + \lceil t/4 \rceil$. Risolvendo iterativamente l'equazione $w_2(t) = t$ si ottiene:

$$t_0 = 2, \quad t_1 = w_2(2) = 5.15, \quad t_2 = w_2(5.15) = 6.15, \quad t_3 = w_2(6.15) = 6.15\,.$$

Dunque nel caso peggiore il job di T_2 ha tempo di esecuzione 6.15 e non rispetterà la propria scadenza relativa uguale a 6.

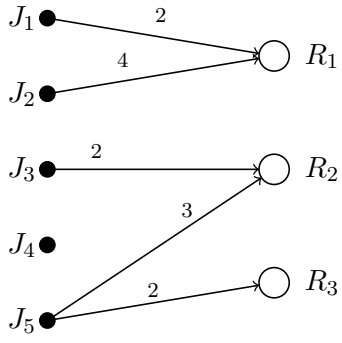
Esercizio 3. *Un sistema deve eseguire cinque job $J_1, \ldots, J_5$, con J_i di priorità maggiore di J_k se $i < k$. I job utilizzano tre risorse condivise R_1, R_2 e R_3, e le relative sezioni critiche sono: $J_1 : [R_1; 2]$, $J_2 : [R_1, 4]$, $J_3 : [R_2; 2]$, $J_4 :$ nessuna, $J_5 : [R_2; 3\,[R_3; 2]\,]$.*

(a) *Quali sono i tempi massimi di blocco dei job per conflitto di risorse utilizzando il procotollo NPCS?*

Il massimo tempo di blocco per conflitto di risorse $b_i(\mathrm{rc})$ del job J_i è dato dalla lunghezza della più lunga sezione critica tra tutti i job di priorità inferiore. Perciò assumendo che i job non si auto-sospendano:

$$b_1(\mathrm{rc}) = 4; \quad b_2(\mathrm{rc}) = 3; \quad b_3(\mathrm{rc}) = 3; \quad b_4(\mathrm{rc}) = 3; \quad b_5(\mathrm{rc}) = 0.$$

(b) *Quali sono i tempi massimi di blocco dei job per conflitto di risorse utilizzando il procotollo priority-inheritance?*



B_d	J_2	J_3	J_4	J_5
J_1	4			
J_2	$*$			
J_3		$*$		3
J_4			$*$	

B_i	J_2	J_3	J_4	J_5
J_1				
J_2	$*$			
J_3		$*$		
J_4			$*$	3

Assumendo che i job non si auto-sospendano, i tempi di blocco per conflitti di risorse di ciascun job sono determinati dal valore massimo su ciascuna riga delle tabelle:

$$b_1(\mathrm{rc}) = 4; \quad b_2(\mathrm{rc}) = 0; \quad b_3(\mathrm{rc}) = 3; \quad b_4(\mathrm{rc}) = 3; \quad b_5(\mathrm{rc}) = 0.$$

(c) *Il tempo di blocco $b_2(rc)$ di J_2 diminuisce nel protocollo priority inheritance rispetto al protocollo NPCS; spiegarne il motivo.*

Nel protocollo NPCS il job J_2 può essere bloccato da qualunque job di priorità inferiore che sta eseguendo una sezione critica; viceversa nel protocollo priority inheritance J_2 può essere bloccato soltanto da un job di priorità inferiore che accede a risorse condivise con J_2 oppure J_1; nell'esempio dell'esercizio non esiste alcun job siffatto.