Sistemi Embedded e Real-time (M. Cesati)

Compito scritto del 15 luglio 2010

Esercizio 1. Si consideri il seguente sistema di task periodici con job non interrompibili: $T_1 = (3, 1)$, $T_2 = (6, 7, 2, 5)$, $T_3 = (6, 2)$.

(a) Determinare una schedulazione ciclica strutturata per il sistema di task che minimizza l'overhead dello scheduler.

(b) Al tempo $t = 35$ viene generato un job sporadico di durata $e = 3$ e scadenza assoluta $t = 115$. È possibile accettare il job assumendo che non esistono altri job sporadici da eseguire? Giustificare la risposta.

Esercizio 2. Si consideri un sistema di task periodici sempre interrompibili e che non si auto-sospendono: $T_1 = (3, \frac{7}{5})$, $T_2 = (6, 1)$, $T_3 = (12, \frac{7}{5})$.

(a) Supponendo che i task sono indipendenti, che venga utilizzato uno scheduler RM, e che l'overhead dello scheduler e del cambio di contesto sia pari a $\mathrm{CS} = \frac{1}{10}$, determinare analiticamente se il sistema è schedulabile su un singolo processore.

(b) Ripetere l'analisi precedente supponendo anche che i job dei task T_1 e T_3 utilizzano una risorsa condivisa ciascuno per un tempo massimo di $\frac{7}{5}$ unità di tempo, e che la risorsa sia controllata tramite il protocollo *priority ceiling*.

Esercizio 3. Si consideri il seguente sistema di task:

$$T_1 = (10, 3),\ T_2 = (8, 3),\ T_3 = (6, 2),\ T_4 = (5, 2),\ T_5 = (25, 6).$$

(a) Determinare analiticamente se l'insieme di task è schedulabile su 4 (**quattro**) processori identici utilizzando l'algoritmo partizionato FFDU.

(b) Determinare il partizionamento dei task sui processori indotto da FFDU.

Sistemi Embedded e Real-time (M. Cesati)

Soluzioni del compito scritto del 15 luglio 2010

Esercizio 1. *Si consideri il seguente sistema di task periodici con job non interrompibili:* $T_1 = (3, 1)$, $T_2 = (6, 7, 2, 5)$, $T_3 = (6, 2)$.

(a) *Determinare una schedulazione ciclica strutturata per il sistema di task che minimizza l'overhead dello scheduler.*

Determiniamo la dimensione del frame f più appropriata:

- Vincolo sulle fasi dei task: il task T_2 ha fase 6, quindi sei deve essere un multiplo intero di f.
- Vincolo sui tempi d'esecuzione dei job:

$$f \geq \max\{1, 2, 2\} = 2$$

- Vincolo sulla divisibilità della lunghezza dell'iperperiodo ($\text{mcm}\{3, 7, 6\} = 42$):

$$f \in \{2, 3, 6\}$$

- Vincolo sul task T_1 ($2f - \gcd\{3, f\} \leq 3$):

$$\begin{aligned} 2 \cdot 2 - \gcd\{3, 2\} &= 3 \leq 3 \\ 2 \cdot 3 - \gcd\{3, 3\} &= 3 \leq 3 \\ 2 \cdot 6 - \gcd\{3, 6\} &= 9 > 3 \quad \Rightarrow f \neq 6 \end{aligned}$$

- Vincolo sul task T_2 ($2f - \gcd\{7, f\} \leq 5$):

$$\begin{aligned} 2 \cdot 2 - \gcd\{7, 2\} &= 3 \leq 5 \\ 2 \cdot 3 - \gcd\{7, 3\} &= 5 \leq 5 \end{aligned}$$

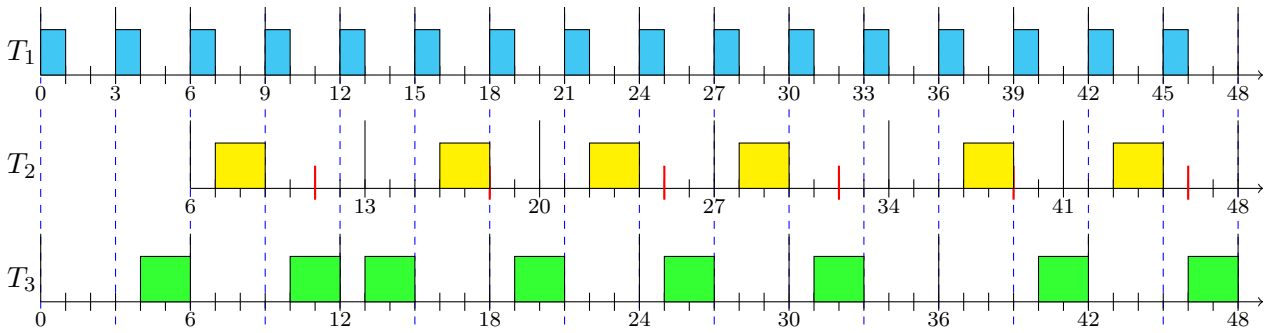
- Vincolo sul task T_3 ($2f - \gcd\{6, f\} \leq 6$):

$$\begin{aligned} 2 \cdot 2 - \gcd\{6, 2\} &= 2 \leq 6 \\ 2 \cdot 3 - \gcd\{6, 3\} &= 3 \leq 6 \end{aligned}$$

Le dimensioni intere ammissibili per il frame sono dunque $f = 2$ e $f = 3$. La dimensione maggiore minimizza l'overhead globale dello scheduler, quindi scegliamo come dimensione del frame il valore tre.

Determiniamo ora una schedulazione ciclica strutturata per l'insieme di task. Notiamo che, poiché T_2 ha fase 6, l'iperperiodo da considerare non è l'intervallo $[0, 42]$ ma piuttosto l'intervallo $[6, 48]$. I due frame nell'intervallo $[0, 6]$ possono perciò essere considerati identici agli ultimi due frame in $[6, 48]$, ovviamente omettendo i job di T_2.

Una possibile soluzione è la seguente:



La schedulazione dei successivi iperperiodi è identica alla schedulazione in $[6, 48]$.

(b) *Al tempo $t = 35$ viene generato un job sporadico di durata $e = 3$ e scadenza assoluta $t = 115$. È possibile accettare il job assumendo che non esistono altri job sporadici da eseguire? Giustificare la risposta.*

Il test di accettazione dei job sporadici utilizzato dall'algoritmo di schedulazione ciclica si basa sul calcolo dello *slack* lasciato libero dai task periodici. Osserviamo che nella schedulazione determinata al punto precedente dell'esercizio esistono solo due unità di tempo lasciate libere dai task periodici in un iperperiodo, precisamente agli istanti di tempo 34 e 35. Poiché la schedulazione si ripete identicamente, gli istanti di tempo a disposizione per i job sporadici rilasciati dopo l'istante 6 sono:

$$34 + 42 \cdot k, \quad 35 + 42 \cdot k \quad (k \geq 0) \quad = \quad 34, 35, 76, 77, 118, 119, 160, 161, \ldots$$

Consideriamo ora il job sporadico rilasciato all'istante $t = 35$. Lo scheduler esegue il test di accettazione solo in corrispondenza dell'inizio del successivo frame, ossia all'istante $t = 36$. Di conseguenza i successivi tre intervalli liberi sono agli istanti 76, 77 e 118. Poiché la scadenza assoluta del job sporadico è all'istante 115, la scadenza non può essere rispettata e lo scheduler deve dunque rifiutare il job sporadico.

Esercizio 2. *Si consideri un sistema di task periodici sempre interrompibili e che non si auto-sospendono:* $T_1 = (3, \frac{7}{5})$, $T_2 = (6, 1)$, $T_3 = (12, \frac{7}{5})$.

(a) *Supponendo che i task sono indipendenti, che venga utilizzato uno scheduler RM, e che l'overhead dello scheduler e del cambio di contesto sia pari a* $CS = \frac{1}{10}$*, determinare analiticamente se il sistema è schedulabile su un singolo processore.*

Osserviamo preliminarmente che le scadenze relative dei task coincidono con i rispettivi periodi. Possiamo dunque cercare di applicare una condizione di schedulabilità basata sul fattore di utilizzazione del sistema di task. Notiamo inoltre che i task sono armonici: per ciascuna coppia di task, uno dei periodi è un multiplo dell'altro. Pertanto, l'algoritmo di schedulazione *Rate Monotonic* è ottimale ed il suo fattore di utilizzazione è uguale a uno.

Dobbiamo però tenere conto dell'overhead introdotto dallo scheduler e dai cambi di contesto, valutato pari a CS = 1/10. Ciò può essere facilmente ottenuto aumentando il tempo di esecuzione dei job di ciascun task in accordo alla formula:

$$e'_i = e_i + 2 \cdot (1 + K_i) \cdot \mathrm{CS}$$

Poiché i job in esame non si autosospendono si ottiene:

$$\begin{aligned} e'_1 &= \frac{7}{5} + 2 \cdot \frac{1}{10} = \frac{8}{5} \\ e'_2 &= 1 + 2 \cdot \frac{1}{10} = \frac{6}{5} \\ e'_3 &= \frac{7}{5} + 2 \cdot \frac{1}{10} = \frac{8}{5} \end{aligned}$$

Il fattore di utilizzazione del sistema è dunque:

$$U_T = \frac{8/5}{3} + \frac{6/5}{6} + \frac{8/5}{12} = \frac{8}{15} + \frac{1}{5} + \frac{2}{15} = \frac{13}{15} \leq 1$$

Perciò il sistema di task è schedulabile con RM.

(b) *Ripetere l'analisi precedente supponendo anche che i job dei task* T_1 *e* T_3 *utilizzano una risorsa condivisa ciascuno per un tempo massimo di* $\frac{7}{5}$ *unità di tempo, e che la risorsa sia controllata tramite il protocollo* priority ceiling.

Determiniamo i tempi di blocco dei vari job dovuti all'accesso alla risorsa condivisa.

Poiché T_1 e T_3 accedono alla stessa risorsa, T_3 può bloccare direttamente l'esecuzione di T_1 per un tempo massimo pari a $\frac{7}{5}$ unità di tempo. Inoltre T_3 può bloccare anche T_2 a causa del *priority inheritance* (se T_1 è bloccato da T_3, la priorità di T_3 supera quella di T_2, che quindi è bloccato da T_1), ancora per un tempo massimo pari a $\frac{7}{5}$ unità di tempo. Di conseguenza, per i tempi di blocco totali $b_i = (1 + K_i) \cdot b_i(rc)$ si ottiene:

$$\begin{aligned} b_1 = (1 + K_1) \cdot b_1(rc) &= \frac{7}{5} \\ b_2 = (1 + K_2) \cdot b_2(rc) &= \frac{7}{5} \\ b_3 &= 0 \end{aligned}$$

Per controllare la schedulabilità del sistema possiamo ancora ragionare sul fattore di utilizzo dei task. Poiché però sono presenti tempi di blocco siamo obbligati ad applicare la condizione di schedulabilità a ciascun task T_i singolarmente, in accordo alla formula:

$$\sum_{k=1}^{i} \frac{e_k}{p_k} + \frac{b_i}{p_i} \leq 1$$

Si noti però che i tempi di esecuzione debbono tenere in considerazione l'overhead dello scheduler e del cambio di contesto, e non coincidono con quelli del precedente punto dell'esercizio. Infatti l'accesso alla risorsa condivisa implica la possibilità di bloccare l'esecuzione se questa dovesse essere occupata, con conseguente doppia invocazione dello scheduler e doppio cambio di contesto aggiuntivi. Pertanto la formula per i tempi di esecuzione relativa ai soli task che accedono ad una risorsa condivisa è:

$$e_i'' = e_i + 4 \cdot (1 + K_i) \cdot \mathrm{CS}$$

Verifica della schedulabilità di T_1:

Poiché $e_1'' = e_1 + 4 \cdot \mathrm{CS} = 7/5 + 4 \cdot 1/10 = 9/5$, la condizione di schedulabilità è:

$$\frac{e_1''}{p_1} + \frac{b_1}{p_1} = \frac{9/5}{3} + \frac{7/5}{3} = \frac{16}{15} > 1$$

Se ne conclude che il task T_1, e dunque l'intero sistema di task, non è più necessariamente schedulabile.

Esercizio 3. *Si consideri il seguente sistema di task:*

$$T_1 = (10, 3),\ T_2 = (8, 3),\ T_3 = (6, 2),\ T_4 = (5, 2),\ T_5 = (25, 6).$$

(a) *Determinare analiticamente se l'insieme di task è schedulabile su 4 (**quattro**) processori identici utilizzando l'algoritmo partizionato FFDU.*

Il fattore di utilizzazione dell'algoritmo partizionato FFDU è:

$$U_{\text{FFDU}} = m \cdot (\sqrt{2} - 1), \quad \text{con } m \text{ pari al numero di processori}$$

quindi per $m = 4$ si ha: $U_{\text{FFDU}} \approx 1.656$.

Il fattore di utilizzazione del sistema di task è:

$$U_T = \frac{3}{10} + \frac{3}{8} + \frac{2}{6} + \frac{2}{5} + \frac{6}{25} = \frac{989}{600} \approx 1.649 \leq U_{\text{FFDU}}$$

Se ne conclude che il sistema di task è schedulabile con FFDU.

(b) *Determinare il partizionamento dei task sui processori indotto da FFDU.*

Indichiamo i quattro processori come P_1, P_2, P_3 e P_4.

Il primo passo dell'algoritmo FFDU consiste nell'ordinare i task per fattore di utilizzazione decrescente:

$$T_4\ (0.4), \quad T_2\ (0.375), \quad T_3\ (0.333), \quad T_1\ (0.3), \quad T_5\ (0.24)$$

Poi si applica la strategia *first fit*:

- T_4: allocato su P_1 ($\{T_4\}$, totale $\frac{2}{5}$)
- T_2: allocato su P_1 ($\{T_2, T_4\}$) perché

  $$\frac{2}{5} + \frac{3}{8} = \frac{31}{40} = 0.775 < 0.828 <= 2(\sqrt{2} - 1) = U_{\text{RM}}(2)$$

- T_3: *non* allocato su P_1 poiché

  $$\frac{31}{40} + \frac{2}{6} = \frac{133}{120} > 1;$$

  quindi è allocato su P_2 ($\{T_3\}$, totale $\frac{1}{3}$)

- T_1: *non* allocato su P_1 poiché

$$\frac{31}{40} + \frac{3}{10} = \frac{43}{40} > 1;$$

allocato su P_2 ($\{T_1, T_3\}$) perché

$$\frac{1}{3} + \frac{3}{10} = \frac{19}{30} < 0.634 < 0.828 < 2(\sqrt{2} - 1) = U_{\mathrm{RM}}(2)$$

- T_5: *non* allocato su P_1 perché

$$\frac{31}{40} + \frac{6}{25} = \frac{203}{200} > 1;$$

non allocato su P_2 perché

$$\frac{19}{30} + \frac{6}{25} = \frac{131}{150} > 0.873 > 0.780 > 3(2^{1/3} - 1) = U_{\mathrm{RM}}(3);;$$

quindi allocato su P_3 ($\{T_6\}$, totale $\frac{6}{25}$)

Se ne conclude che FFDU determina il seguente partizionamento:

$$\begin{array}{lcl} P_1 & : & \{T_2, T_4\} \\ P_2 & : & \{T_1, T_3\} \\ P_3 & : & \{T_5\} \\ P_4 & : & \text{scarico} \end{array}$$

Si osservi che tipicamente la schedulabilità di un task su un processore in un algoritmo *first fit* è stabilita analizzando una condizione di schedulabilità (condizione sufficiente), piuttosto che eseguendo un test di schedulabilità con la funzione di tempo necessario. Infatti il test è molto oneroso dal punto di vista computazionale: oltre ad effettuare il test del task candidato per l'inserimento considerando tutti i task di priorità (frequenza) superiore, è necessario anche controllare che tutti i task di priorità (frequenza) inferiore continuano ad essere schedulabili considerando anche il task candidato.