

# Gestione della Preemption nei sistemi a microcontrollore

di Paolo Gai e Giorgio Buttazzo

Questo articolo descrive le principali caratteristiche delle tecniche di scheduling preemptive e non preemptive utilizzabili nei dispositivi a microcontrollore con il sistema operativo ERIKA Enterprise ed in generale con sistemi operativi di classe OSEK.

La preemption è la possibilità da parte di un sistema di interrompere una attività in corso per eseguirne un'altra in quel momento giudicata "più importante". Esempi di preemption sono l'interruzione dell'esecuzione del codice di un microcontrollore a causa dell'arrivo di un interrupt, il termine di un quanto di tempo in un sistema operativo time-sharing, oppure la preemption dovuta all'attivazione di un task ad alta priorità in un sistema operativo real-time.

I sistemi di elaborazione embedded possono in generale essere suddivisi in due categorie fondamentali: i sistemi preemptive ed i sistemi non preemptive. Nei sistemi preemptive, l'attività in esecuzione può essere sempre interrotta da altre attività giudicate più importanti. Molti sistemi operativi real-time implementano tale meccanismo in quanto, come vedremo più avanti nel corso di questo articolo, la preemption migliora i tempi di risposta delle attività rendendo il sistema più semplice da schedare. In un sistema non preemptive, l'esecuzione di una attività non può essere interrotta una volta iniziata.

La Figura 1 mostra il comportamento delle due tipologie di sistemi in seguito all'arrivo di una attività "importante". Nel caso di un sistema preemptive, il task a bassa priorità task2 viene interrotto per essere ripreso successivamente al termine dell'attività task1. Nel caso di un sistema non pre-

emptive, l'attività task2 non viene interrotta, ritardando in qualche modo l'attività task1. In figura, tale ritardo è evidenziato come "tempo di bloccaggio", definito come la quantità di tempo che un task è costretto ad attendere a causa della esecuzione di una attività a priorità più bassa. In generale, il tempo di bloccaggio è un effetto collaterale della non preemption che limita le prestazioni temporali di un sistema embedded. Ad oggi, molti sistemi embedded sono programmati in modo non preemptive. Come esempio, possiamo citare la classica implementazione dei sistemi real-time che utilizza un unico ciclo infinito di attesa su eventi asincroni. In tali sistemi sono spesso presenti degli interrupt utilizzati per notificare tali eventi asincroni che tipicamente effettuano preemption sulle attività applicative per una quantità di tempo trascurabile. Esclusa tale sorgente di preemption, le restanti attività applicative sono gestite all'interno di un ciclo infinito in modo sequenziale (o prioritario), per cui non è possibile controllare prima del termine dell'esecuzione corrente, l'arrivo di eventi che richiedono azioni urgenti. La programmazione preemptive è tipicamente utilizzata all'interno dei sistemi operativi general purpose e nei sistemi real-time di tipo più avanzato. Ad esempio, i sistemi operativi per PC sono tutti di tipo preemptive, e gestiscono il tempo della CPU suddividendolo tra i vari proces-

si “pronti” in un determinato istante. Al termine di un quanto di tempo predefinito (tipicamente oscillante da 10 a 50 ms), il sistema effettua una preemption per eseguire un altro task pronto.

Le due sezioni che seguono hanno lo scopo di presentare brevemente i vantaggi e gli svantaggi dei due approcci.

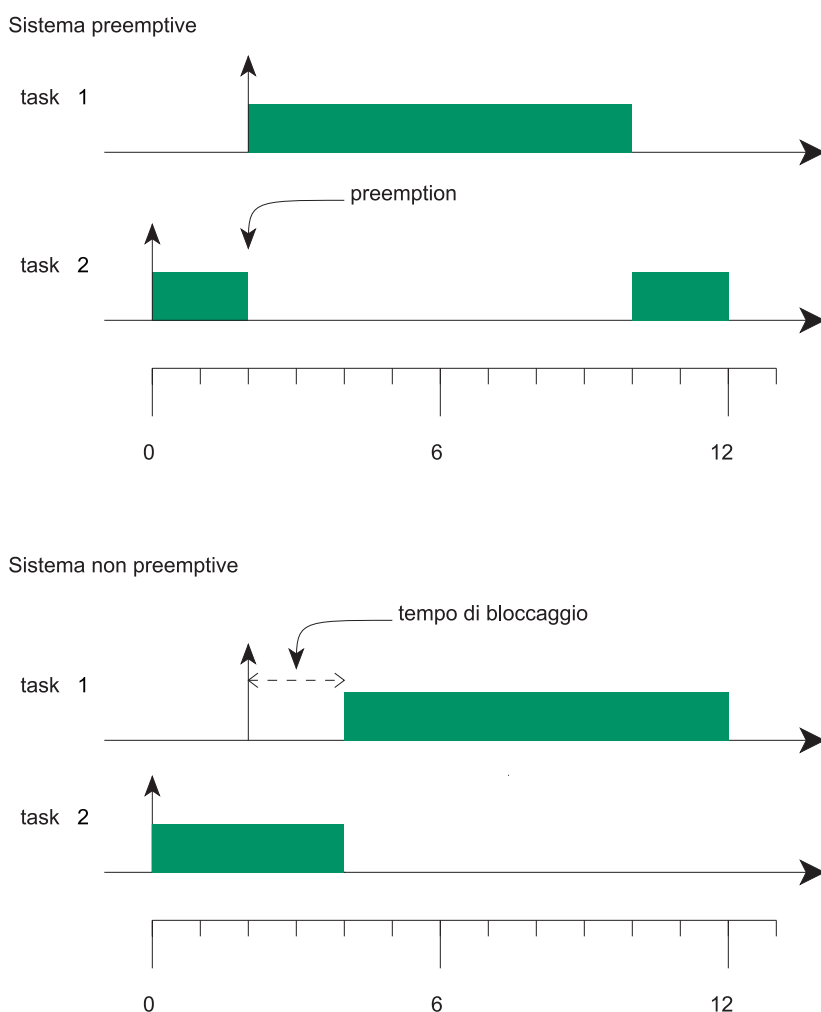
### VANTAGGI E SVANTAGGI DELLA SCHEDULAZIONE NON PREEMPTIVE

Uno dei maggiori vantaggi della schedulazione non preemptive consiste nell’evitare l’accesso concorrente a risorse condivise, semplificando la gestione della concorrenza, in quanto per costruzione non può mai succedere che un’attività venga interrotta nel mezzo della manipolazione di strutture dati complesse. Come conseguenza diretta, i sistemi non preemptive non devono sopportare l’overhead delle primitive di sistema operativo che gestiscono le risorse (anzi, spesso tali sistemi sono implementati senza un vero e proprio sistema operativo). Un altro vantaggio dei sistemi non preemptive è quello di

richiedere poca memoria RAM per la gestione dello stack, in quanto in uno stesso istante non ci può essere più di una attività in esecuzione che ha allocato spazio di stack per le proprie funzioni. Lo svantaggio principale della programmazione non preemptive è il fatto che attività ed eventi considerati importanti devono attendere la terminazione dell’evento in corso per essere gestiti. Un secondo problema è legato al fatto che non è possibile realizzare attività applicative “troppo lunghe” per non costringere le attività importanti ad attendere troppo. In tal caso, l’attività complessa dovrà essere suddivisa in attività più brevi con conseguente complicazione e bassa leggibilità del codice applicativo. Dal punto di vista formale, inoltre, un sistema non preemptive ha lo svantaggio di poter risultare non schedabile anche in casi in cui l’utilizzazione del processore sia arbitrariamente bassa.

### VANTAGGI E SVANTAGGI DELLA PROGRAMMAZIONE PREEMPTIVE

Tra i vantaggi della programmazione preemptive, possiamo ricordare la maggiore prontezza di risposta del sistema agli eventi esterni, in quanto le attività importanti non devono attendere il termine di attività a più bassa priorità. Come vedremo nella sezione successiva, ciò si traduce in una migliore schedabilità del sistema. Inoltre, i sistemi preemptive sono tipicamente più semplici da programmare in quanto la scrittura del codice è indipendente dal comportamento degli altri thread e non dipende dalla durata di esecuzione del codice: in particolare, non è necessario dividere le attività in parti piccole indipendenti per limitare i tempi di bloccaggio dei task ad alta priorità. Uno svantaggio della schedulazione preemptive è la necessità di utilizzare un sistema operativo che introduce un maggiore overhead sull’esecuzione dei processi e che richiede un spazio aggiuntivo di memoria RAM per la gestione degli stack allocati ad ogni thread. Tuttavia, tali svantaggi possono essere mitigati grazie all’utilizzo di sistemi operativi opportuni,



**Figura 1**  
Differenza tra preemption e bloccaggio

appositamente studiati per i microcontrollori, e tramite l'utilizzo di tecniche di ottimizzazione e condivisione dello stack (questa tematica verrà analizzata in uno dei prossimi articoli).

### TECNICHE DI GARANZIA PER TASK NON PREEMPTIVE

Questa sezione analizza le tecniche utilizzabili per fornire delle garanzie temporali nell'esecuzione di task non preemptive. In generale, tali task possono essere costituiti da funzioni richiamate all'interno del ciclo infinito `main()` oppure da attività di sistema configurate opportunamente. Nel primo caso, occorre sottolineare come, al fine di ridurre i tempi di risposta delle attività più importanti, i task da eseguire devono essere eseguiti in ordine prioritario, ad esempio utilizzando l'istruzione `continue` nel seguente modo:

```
int main(int argc, char **argv)
{
    ...
    for (;;) {
        if (evento1) {
            gestione_evento1();
            continue;
        }
        if (evento2) {
            gestione_evento2();
            continue;
        }
        ...
    }
    return 0;
}
```

Dato un insieme di task con vincoli temporali (ad esempio, esecuzioni periodiche per il campionamento di un sensore), sarebbe desiderabile stabilire a priori se tali vincoli possano essere rispettati. Al fine di analizzare la fattibilità della schedulazione, la non preemption può essere considerata come un caso particolare di utilizzo delle risorse condivise. In particolare, i task non preemptive si comportano come se eseguissero tutta la loro computazione in una sezione critica comune. Utilizzando le tecniche di analisi presentate nell'articolo pubblicato su Firmware di Marzo 2007, un insieme di  $N$  task non preemptive,  $T_1 \dots T_N$ , con tempi di calcolo  $C_1 \dots C_N$ , e periodi  $\tau_1 \dots \tau_N$ , ordinati per periodo crescente (il task  $\tau_1$  è quello con periodo più basso), risulta schedulabile se, per ogni task  $i$ , vale la formula:

$$\forall i, 1 \leq i \leq n, \sum_{k=1}^i \frac{C_k}{T_k} + \frac{B_i^{max}}{T_i} \leq \ln 2$$

L'interpretazione della formula è la seguente: il primo termine considera il fattore di utilizzazione del processore dei task a più alta priorità del task considerato.  $B_i^{max}$  rappresenta il tempo massimo di bloccaggio che il task può subire a causa della non preemption. Tale fattore è calcolato come il massimo tempo di esecuzione dei task a priorità più bassa del task considerato, ovvero:

$$B_i^{max} = \max_{k=i+1..n} (C_k)$$

Il fattore  $\ln 2$  è il limite di schedulabilità di Liu & Layland presentato nell'articolo pubblicato su Firmware n. 13 (Febbraio 2007).

Confrontando tale formula con quella presentata nel precedente articolo, che riportiamo di seguito per semplicità:

$$\sum_{i=1}^N \frac{C_i}{T_i} \leq \ln 2$$

è possibile notare che il fattore:

$$\frac{B_i^{max}}{T_i}$$

riduce la schedulabilità di un sistema non preemptive. Ad esempio, la formula implica che in un sistema non preemptive non è possibile avere task di durata superiore al periodo del task più frequente. Ciò costringe gli sviluppatori a spezzare le attività troppo lunghe in un insieme di attività più brevi. Tale limitazione non è in generale presente nei sistemi preemptive.

La Figura 2 mostra un caso in cui un task *task1* ad alta priorità (con periodo di 10 unità) deve attendere *task2* a priorità più bassa (con periodo di 25 unità) rischiando di non rispettare i propri vincoli temporali.

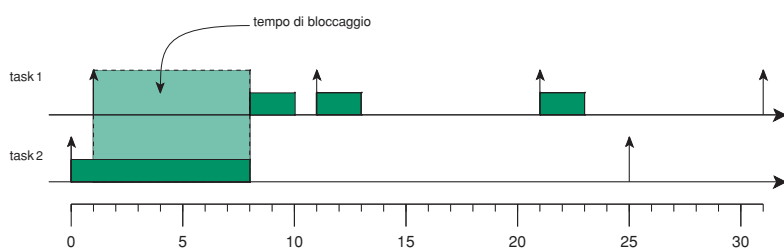
Infine, è utile segnalare che non esiste un estremo superiore di utilizzazione del processore (come ad esempio il fattore  $\ln 2$  nel caso preemptive) al di sotto del quale la schedulabilità è sempre garantita. In altre parole, è sempre possibile trovare un insieme di task non schedulabili con utilizzazione arbitrariamente bassa. Ad esempio, il task set composto da due task  $\tau_1$  e  $\tau_2$  con priorità  $P_1 > P_2$ , utilizzazione  $C_1/T_1 = \varepsilon$  arbitrariamente bassa,  $C_2 > T_1$ ,  $C_1 = \varepsilon T_1$ ,  $T_2 = C_2/\varepsilon$ , non è schedulabile pur avendo una utilizzazione del processore arbitrariamente bassa.

## NON PREEMPTION NEI SISTEMI OPERATIVI OSEK/VDX

La necessità di avere schedulazioni non preemptive nei sistemi real-time per microcontrollori è talmente sentita da essere stata incorporata nello standard OSEK/VDX, nonché nel kernel real-time open-source ERIKA Enterprise Basic GPL. Come citato nell'articolo di Marzo, la non preemption può essere ottenuta nei sistemi OSEK in due modi: tramite il locking di una apposita risorsa denominata `RES_SCHEDULER`, oppure tramite l'utilizzo di task non preemptive. Nel secondo caso, occorre dichiarare i task come non preemptive, includendo nel file OIL una riga "`SCHEDULE = FULL;`" per ciascun task. In questo caso, l'intero codice del task viene eseguito in modo non preemptive, a meno di una chiamata della primitiva `schedule()`, che implementa un punto di preemption all'interno dei task. In seguito alla chiamata di tale primitiva, un eventuale task pronto in coda ready a priorità più alta di quello in esecuzione effettuerà preemption sul task in esecuzione. Nel caso di scheduling preemptive, la primitiva `schedule()` non ha alcun effetto.

## NON PREEMPTION ED ALGORITMI DI CONTROLLO

Molti sistemi di controllo industriale sono costituiti da un insieme di task che hanno una struttura tipica comprendente una parte iniziale di acquisizione dei sensori, una fase di calcolo (che realizza l'algoritmo di controllo) e una fase di output che pilota gli attuatori. In molti sistemi di controllo, tra i fattori che influiscono sulla stabilità del sistema sono da considerare l'intervallo di tempo che intercorre tra l'ingresso e l'uscita dei dati (denominato input-output delay) e la sua variabilità (denominata input-output jitter). In generale, tali intervalli possono variare a causa delle preemption subite dai task in un sistema preemptive. Lo scheduling non preemptive può essere utilizzato come un metodo per la riduzione sia del delay che del jitter di un task di controllo. Tale metodo offre i seguenti vantaggi:



**Figura 2**  
il task task1 rischia di non rispettare i propri vincoli temporali a causa del task task2 e dello scheduling non preemptive

- il jitter di input-output è ridotto al minimo possibile per tutti i task del sistema (supponendo che i tempi di esecuzione dei task siano costanti nel tempo);
- l'input-output delay è costante ed è uguale al tempo di calcolo del task. Ciò è rilevante in quanto molti sistemi di controllo sono più sensibili al delay che al jitter.

## PREEMPTION THRESHOLDS

La tecnica dei preemption threshold fornisce una soluzione di compromesso fra l'approccio preemptive e quello non preemptive. Essa è stata inizialmente proposta da Saksena e Wang] e consiste nell'assegnare a ciascun task due priorità distinte, chiamate ready priority e threshold. La ready priority è utilizzata durante l'accodamento del task nella coda ready del sistema operativo, mentre il threshold è una priorità maggiore o uguale alla ready priority che il task acquisisce quando viene messo in esecuzione. Di fatto, è come se il task eseguisse ad una priorità "aumentata" rispetto a quella utilizzata per l'accodamento, o, in altre parole, è come se il task effettuasse il lock di una speciale risorsa per tutto il suo tempo di esecuzione, utilizzando il protocollo Immediate Priority Ceiling. Utilizzando tale tecnica è anche possibile ottenere un sistema totalmente preemptive o non preemptive, come caso particolare. Infatti, se per ogni task la ready priority è uguale al threshold, si ottiene un sistema totalmente preemptive. Se invece tutti i task hanno threshold uguale alla più alta priorità esistente, il sistema diventa non preemptive. In generale, quello che ci si può aspettare è che il sistema si comporti in modo intermedio tra i due casi estremi. Inoltre, è stato dimostrato che esistono insiemi di task schedulabili con Preemption Thresholds ma che non lo sono né con una schedulazione preemptive, né con una schedulazione non preemptive. Pertanto, il metodo del preemption threshold rappresenta una categoria di scheduling a sé. La schedulazione con i preemption threshold è stata recepita anche nello standard OSEK tramite l'utilizzo delle Internal Resources, che rappresentano delle speciali risorse (non utilizzabili dall'utente)

che vengono automaticamente bloccate e rilasciate quando un task va in esecuzione. L'utilizzo dei preemption threshold, come vedremo in uno dei prossimi articoli, trova il suo principale utilizzo nei sistemi operativi real-time per microcontrollori, al fine di ridurre la quantità di memoria RAM dedicata allo stack dei processi.

**Codice MIP 015044**