

# Capitolo 1

## Algoritmi di Scheduling

### 1.1 Sistemi Real Time

In un sistema in *tempo reale* (real time) il tempo gioca un ruolo essenziale. Le applicazioni di tali sistemi sono molteplici e di larga diffusione. Ad esempio, un dispositivo lettore di compact disc, prende i bit in uscita dal drive e li converte in musica. Questa operazione deve compiersi in un intervallo di tempo ristretto, in quanto eventuali ritardi determinerebbero alterazioni nella riproduzione del suono. Sistemi di monitoraggio (ad esempio in una struttura ospedaliera) e di controllo (il pilota automatico di un aereo, il controllo robotizzato in una fabbrica, etc.) sono altri esempi comuni di sistemi che devono operare in tempo reale. In tutti questi casi, avere la risposta corretta, ma in ritardo, è come non averla affatto se non peggio.

I sistemi real time sono normalmente classificati in accordo a vincoli temporali che possono essere più o meno stringenti. In particolare:

- sistemi *hard real time*: sono sistemi che devono rispettare scadenze temporali in modo incondizionato. Non rispettare anche una sola scadenza può avere ripercussioni gravi sull'intero sistema;
- sistemi *soft real time*: sistemi in cui non rispettare occasionalmente una scadenza non è opportuno ma può essere tollerato.

In entrambi i casi, il comportamento real time è ottenuto dividendo un programma in processi il cui comportamento è prevedibile e noto in anticipo. Questi processi hanno normalmente una vita breve e possono essere eseguiti nella loro completezza in un tempo inferiore al secondo. Nel momento in cui viene rilevato un evento esterno, lo scheduler dei processi ha il compito di determinare un'opportuna sequenza di esecuzione dei processi tale da garantire il rispetto di tutte le scadenze.

Gli eventi a cui un sistema real time deve poter reagire possono essere classificati in:

- *periodici*: si verificano ad intervalli di tempo regolari;
- *aperiodici*: si verificano in modo imprevedibile.

Nel caso di eventi periodici, se esistono  $m$  eventi e se l'evento  $i$  arriva con periodo  $P_i$  e richiede  $C_i$  secondi di tempo di CPU per essere gestito, il carico può essere gestito solo se:

$$\sum_{i=1}^m \frac{C_i}{P_i} \leq 1. \quad (1.1)$$

Un sistema real time che rispetta questo vincolo è detto *schedulabile*. Ad esempio, consideriamo un sistema soft real time con tre eventi periodici di periodo 100, 200 e 500 millisecondi, rispettivamente. Se questi eventi richiedono rispettivamente 50, 30 e 100ms di tempo di CPU per la loro esecuzione, il sistema è schedulabile in quanto:  $50/100 + 30/200 + 100/500 = 0,5 + 0,15 + 0,2 \leq 1$ . Se un quarto evento, con periodo di 1 secondo, si aggiunge al sistema, l'insieme dei processi rimarrà schedulabile fino a quando questo evento non richiederà più di 150ms di tempo di CPU per occorrenza. Notare che nel calcolo precedente si suppone implicitamente che l'overhead per il cambio di contesto sia così piccolo da poter essere trascurato.

### 1.1.1 Scheduling dei Sistemi Real Time

Gli algoritmi di scheduling real time possono essere distinti in:

- *statici*: la decisione di schedulazione è presa prima che il sistema inizi l'esecuzione dei processi. Questi metodi richiedono che le informazioni complete circa il lavoro da fare e le scadenze da rispettare siano disponibili in anticipo rispetto all'esecuzione dei processi;
- *dinamici*: la decisione di schedulazione è presa durante l'esecuzione dei processi. Non hanno restrizioni circa la conoscenza anticipata sui tempi di esecuzione e le scadenze da rispettare.

Nel seguito verranno analizzate alcune politiche di scheduling real time, facendo riferimento al particolare contesto costituito dalle applicazioni multimediali. Infatti, i sistemi operativi che supportano applicazioni multimediali, differiscono da quelli tradizionali per tre aspetti principali: la schedulazione dei processi, il file system e la schedulazione del disco.

#### Schedulazione di processi omogenei

È la situazione che si presenta quando più processi con uguali richieste e vincoli temporali devono essere serviti in modo efficiente dalla politica di scheduling. Ad esempio, una tale situazione si presenta per un server video che deve supportare la visualizzazione di un numero fisso di video tutti

caratterizzati dalla stessa frequenza dei frame (*frame rate*), risoluzione video, frequenza di trasmissione dati, etc. In questa situazione una semplice ma efficace politica di scheduling è il round-robin. Infatti, tutti i processi sono ugualmente importanti, hanno la stessa quantità di lavoro da svolgere e si bloccano quando hanno terminato l'elaborazione del frame corrente. L'algoritmo di schedulazione può essere ottimizzato aggiungendo un meccanismo di temporizzazione per assicurare che ogni processo sia eseguito alla frequenza corretta.

### Schedulazione generale in tempo reale

Il semplice modello precedente si presenta raramente nella pratica. Un modello più realistico prevede la presenza di più processi che competono per l'uso della CPU, ciascuno con il proprio carico di lavoro e le proprie scadenze temporali. Nel seguito supporremo che il sistema conosca la frequenza con cui eseguire ciascun processo, quanto lavoro debba compiere ogni processo e la successiva scadenza temporale.

Come esempio di ambiente in cui lavori uno schedulatore multimediale in tempo reale si considerino i tre processi *A*, *B* e *C* di Fig.1.1. Il processo *A* viene eseguito ogni  $30ms$  e ogni frame richiede  $10ms$  di tempo di CPU. In assenza di competizione sarebbe eseguito nei periodi  $A_1$ ,  $A_2$ ,  $A_3$  etc., ciascuno  $30ms$  dopo il precedente. Ogni periodo di CPU gestisce un frame e ha una scadenza, cioè deve terminare prima che inizi il successivo. I processi *B* e *C* in Figura sono eseguiti rispettivamente 25 e 20 volte al secondo, con tempi di calcolo di  $15ms$  e  $5ms$ .

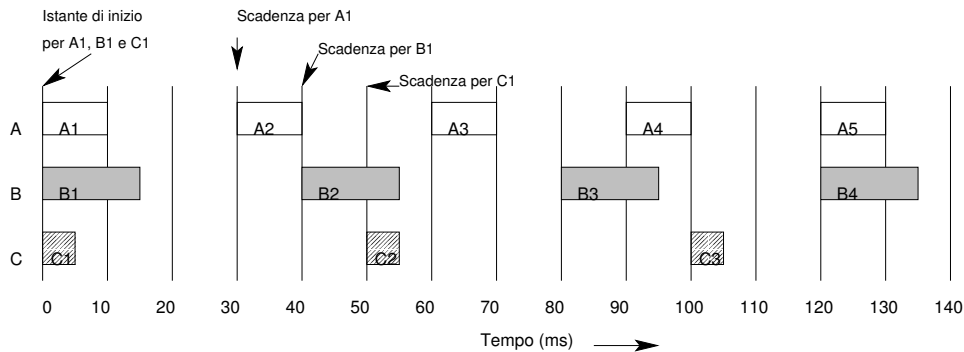


Figura 1.1: Tre processi periodici che visualizzano ciascuno un film. Le frequenze dei frame e i tempi di elaborazione per frame sono diversi per ciascun film.

Il problema diventa quello di schedulare *A*, *B* e *C* per essere certi che rispettino ciascuno le proprie scadenze temporali. Prima di cercare un algoritmo di schedulazione, è necessario valutare se questo insieme di processi sia effettivamente schedulabile. A questo fine è possibile utilizzare l'Eq.(1.1), che

nel caso dei processi  $A$ ,  $B$  e  $C$  dell'esempio produce:  $10/30 + 15/40 + 5/50 = 0.808$  del tempo di CPU, ed il sistema dei processi è quindi schedulabile.

Esistono sistemi in tempo reale in cui i processi possono subire o meno prelazione. Nei sistemi multimediali i processi sono generalmente prelazionabili: un processo la cui scadenza temporale sia a rischio può interrompere il processo in esecuzione prima che esso completi l'elaborazione del proprio frame; quando ha terminato, il processo precedente può continuare. In aggiunta, gli algoritmi di schedulazione in tempo reale possono essere sia *statici* che *dinamici*. Quelli statici assegnano a ciascun processo una priorità determinata in precedenza ed effettuano una schedulazione con prelazione e con priorità utilizzando le priorità stesse. Gli algoritmi dinamici non hanno priorità prefissate.

**Schedulazione a frequenza monotona** Il classico algoritmo statico di schedulazione in tempo reale per processi periodici e prelazionabili è RMS (*Rate Monotonic Scheduling*, schedulazione a frequenza monotona). È utilizzabile per processi che soddisfano le seguenti condizioni:

1. ogni processo periodico deve essere completato entro il suo periodo di tempo;
2. nessun processo è dipendente dagli altri;
3. ogni processo necessita della stessa quantità di tempo di CPU per ogni periodo di esecuzione;
4. i processi non periodici non hanno scadenze temporali;
5. la prelazione dei processi avviene istantaneamente e senza sovraccarico di lavoro per il sistema.

Le prime quattro condizioni sono ragionevoli, mentre l'ultima rende più semplice la modellazione del sistema.

RMS assegna a ciascun processo una priorità prefissata uguale alla frequenza con cui deve essere eseguito. Ad esempio, un processo che debba essere eseguito ogni  $30ms$  (33 volte/s) acquisisce priorità 33; un processo da eseguire ogni  $40ms$  (25 volte/s) acquisisce priorità 25, mentre un processo da eseguire ogni  $50ms$  (20 volte/s) acquisisce priorità 20. Dato che le priorità variano linearmente con la frequenza (numero di volte al secondo in cui il processo è eseguito), il metodo è detto a frequenza monotona. Durante l'esecuzione, lo schedulatore esegue sempre il processo pronto a più alta priorità, prelazionando, se necessario, il processo in esecuzione. È stato dimostrato che RMS è ottimale rispetto alla classe di algoritmi di schedulazione statici. La Fig.1.2 illustra il funzionamento dell'algoritmo di schedulazione a frequenza monotona relativamente ai processi dell'esempio di Fig.1.1. I processi

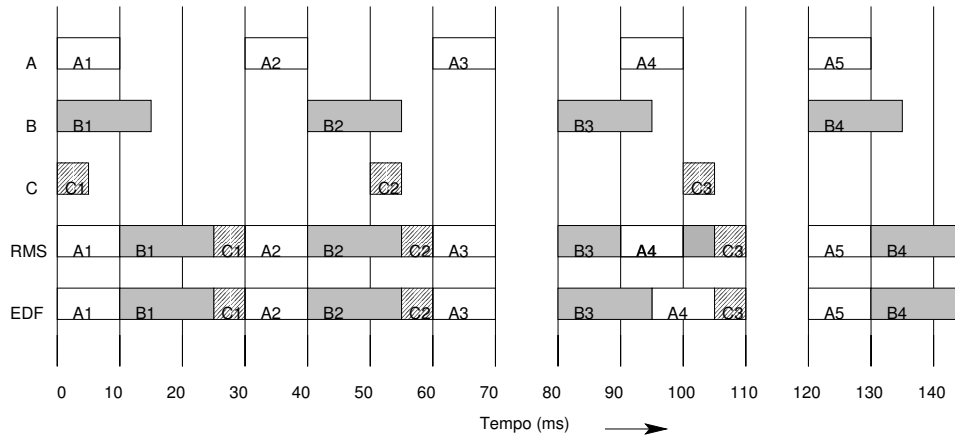


Figura 1.2: Esempio di schedulazione in tempo reale con RMS e EDF.

$A$ ,  $B$  e  $C$  hanno rispettivamente priorità statiche 33, 25 e 20. Come conseguenza, quando  $A$  deve andare in esecuzione prelaiona ogni altro processo;  $B$  può prelaionare  $C$ , mentre  $C$  per andare in esecuzione deve attendere fino a quando la CPU non è libera.

**Schedulazione con priorità alla scadenza più vicina** L'algoritmo EDF (*Earliest Deadline First*, schedulazione con priorità alla scadenza più vicina), è un algoritmo dinamico e pertanto non richiede né che i processi siano periodici, né che abbiano lo stesso tempo di esecuzione per periodo di CPU. Con questo approccio, è sufficiente che un processo che ha bisogno della CPU annunci la sua presenza e la scadenza temporale. Lo schedulatore mantiene una lista dei processi eseguibili, ordinata rispetto alla scadenza temporale; l'algoritmo esegue il primo processo della lista, cioè quello con scadenza temporale più vicina. Quando un nuovo processo è pronto, il sistema controlla se la sua scadenza preceda quella del processo correntemente in esecuzione; in caso affermativo il nuovo processo prelaiona quello corrente. La Fig.1.2 presenta un esempio di schedulazione con EDF. Un secondo esempio che confronta RMS e EDF è mostrato in Fig.1.3.

È interessante notare come nell'esempio riportato in Fig.1.3 l'algoritmo RMS fallisca. Questo è dovuto al fatto che, utilizzando priorità statiche, l'algoritmo funziona solo se l'utilizzo della CPU non è troppo elevato. È possibile dimostrare che, per ogni sistema di processi periodici, se:

$$\sum_{i=1}^m \frac{C_i}{P_i} \leq m \cdot (2^{1/m} - 1) \tag{1.2}$$

allora è garantito il funzionamento di RMS. Per  $m$  uguale a 3, 4, 5, 10, 20, 100 le massime utilizzazioni permesse sono 0.780, 0.757, 0.743, 0.718, 0.705 e 0.696. Per  $m$  che tende all'infinito, l'utilizzo massimo della CPU tende

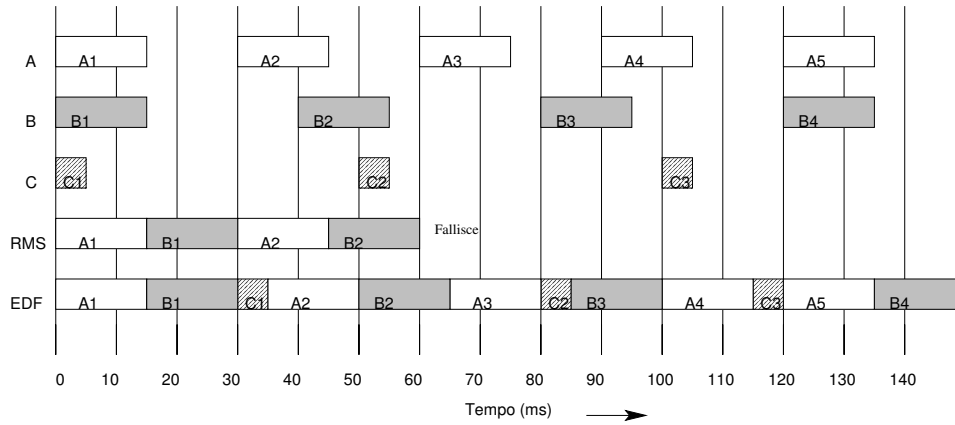


Figura 1.3: Esempio di schedulazione in tempo reale con RMS e EDF.

in modo asintotico a  $\ln 2$ . Questo significa che per  $m = 3$ , RMS funziona sempre se l'utilizzazione della CPU è uguale o minore di 0.780. Nell'esempio di Fig.1.3 l'utilizzo della CPU, calcolato con l'Eq.(1.1), è così elevato (0.975) da non permettere il funzionamento di RMS.

Al contrario, EDF funziona sempre per qualunque insieme di processi schedulabile e può raggiungere il 100% di utilizzo della CPU. Il prezzo di questo è pagato in termini di una maggiore complessità dell'algoritmo EDF rispetto a RMS.