Università degli studi di Modena e Reggio Emilia

Dipartimento di Ingegneria Enzo Ferrari

Real Time Embedded System

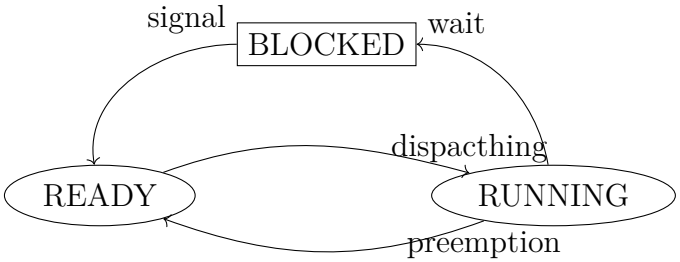
Anno Accademico 2023/24

Indice

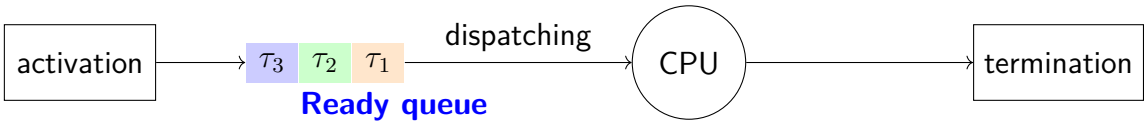
Capitolo 1

Introduzione

Task: è un insieme di sequenze di istruzioni, che in assenza di altre attività, vengono continuamente eseguite dal processore finché non vengono completate.
Può essere un processo o un thread in base al sistema operativo.



Ready Queue: i task "pronti" (*ready*) sono contenuti all'interno di una coda di attesa, anche nota come *ready queue*. La strategia con cui vengono scelti i task dalla coda per essere eseguiti sulla *CPU* sono gli ***scheduling algorithms***.

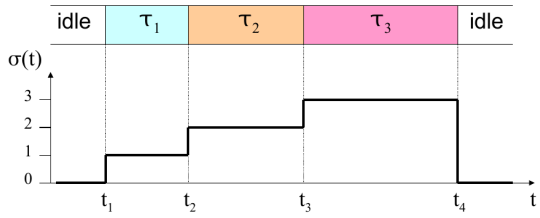


Scheduling può essere definito ***preemptive*** ovvero se il task in esecuzione in un certo istante di tempo t_i può essere temporaneamente sospeso per eseguire un task con importanza maggiore, mentre si dice ***non-preemptive*** se il task in esecuzione non può essere sospeso finché non viene completato.

Schedule: uno *schedule* è un particolare assegnamento di task ad un processore. Dato un ***taskset*** $\mathcal{T} = \{\tau_1, ..., \tau_n\}$ uno *schedule* viene mappato to σ:

$$\mathbb{R}^+ \to \mathbb{N} \mid \forall t \in \mathbb{R}^+ \qquad \sigma(t) = \begin{cases} k > 0 & \text{if } \tau_k \text{ is running} \\ 0 & \text{if the processor is idle} \end{cases}$$

Consideriamo il *task set*: $\{\tau_1, \tau_2, \tau_3\}$



Nei punti t_1, t_2, t_3 e t_4 viene eseguito un ***content switch***, ogni intervallo di tempo $[t_i, t_{i+1})$ viene chiamato ***time slice***.

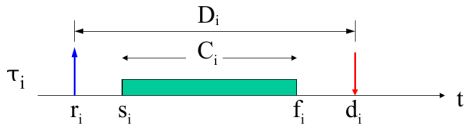


Figura 1.1: Real-time tasks

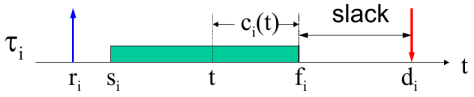


Figura 1.2: Real-time tasks

- $\mathbf{r}_i$ è il ***request time***.
- $\mathbf{s}_i$ è lo ***start time*** ovvero il tempo in cui il task inizia l'esecuzione.
- $\mathbf{C}_i$ è il tempo di esecuzione in caso peggiore (***WCET***).
- $\mathbf{d}_i$ è la ***deadline*** **assoluta**, mentre $\mathbf{D}_i$ è la ***deadline*** **relativa**.
- $\mathbf{f}_i$ è il ***finishing time*** ovvero il tempo effettivo in cui il task completa il suo lavoro
- ***lateness***: $L_i = f_i - d_i$, è quindi la differenza tra il tempo di fine del task e la sua deadline assoluta, se ≤ 0 allora il task ha rispettato la sua deadline se no la deadline è stata missata [***tardiness***: $max(0, L_i)$]

- ***Residual WCET***: $c_i(t)$
- ***laxity (o slack)***: $d_i - t - c_i(t)$

Tasks vs. Jobs: un task è un infinita sequenza di istanze che vengono ripetute [*jobs*]. È possibile differenziare varie tipologie di *task* in base a quale deve essere la loro garanzia di rispetto delle loro *deadline*:

1. ***Hard Task***: tutti i *jobs* devono rispettare le proprie deadline, mancare una deadline comporta serie conseguenze.
2. ***Firm Task***: solo alcuni *jobs* possono missare la loro deadline.
3. ***Soft Task***: i *jobs* possono missare la loro deadline, l'obiettivo è quello di massimizzare la ***responsiveness***.

Un sistema operativo capace di gestire *hard task* viene chiamato ***hard real-time system***. I *tasks* possono avere due modalità di **attivazione**:

1. ***time driven***: anche noti come **tasks periodici**, i task vengono automaticamente attivati dal *kernel* ad intervalli regolari. Definiamo il task come: $\tau_i(C_i, T_i, D_i)$ dove $\mathbf{T_i}$ è il periodo a cui quel task viene invocato.

$$\begin{cases} r_{i,k} = \Phi_i + (k+1) \cdot T_i \qquad k = 1 \rightarrow r_{i1} = \Phi_i \\ d_{i,k} = r_{i,k} + D_i \end{cases}$$

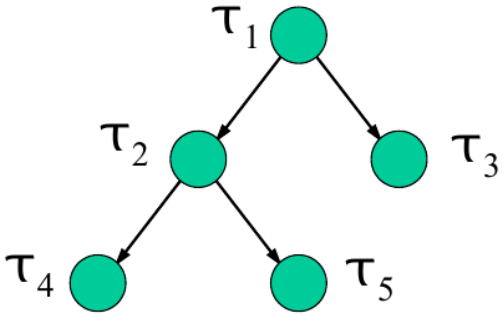
2. ***event driven***: anche noti come **tasks aperiodici**, ovvero il task viene attivato all'arrivo di un evento o per un'invocazione esplicita della sua primitiva di invocazione. A loro volta possono dividersi in:
   - ***aperiodic***: $r_{i,k+1} > r_{i,k}$
   - ***sporadic***: $r_{i,k+1} \geq r_{i,k} + T_i$

Sui *tasks* possono essere imposti dei vincoli, che si differenziano in:

- ***timinig constraints***: ovvero dei vincoli sul tempo di esecuzione [*deadline*, *activation*, *completition* e *jitter*], possono essere **impliciti** o **espliciti**:

  - ***explicit constraints***: sono definite nelle specifiche del sistema di attivazione: apertura della valvola ogni 10s
  - ***implicit constraints***: non appaiono nelle specifiche direttamente, ma devono essere rispettate per seguire i vincoli di utilizzo del sistema: schivare ostacoli mentre si corre ad una velocità v.

- ***precedence constraints***: alcuni task devono rispettare delle precedenze di esecuzione, normalmente specificate da un ***Directed Acyclic Graph***:



predecessore

$\tau_1 \prec \tau_4$

predecessore immediato

$\tau_1 \rightarrow \tau_2$

- ***resource constraints***: per preservare *data consistency* bisogna accedere alle risorse condivise in **mutua esclusione**, che però introduce un *delay*.

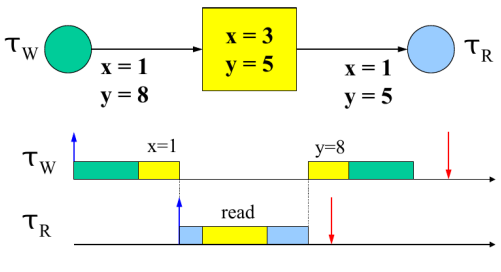


Figura 1.3: *no mutual exclusion*

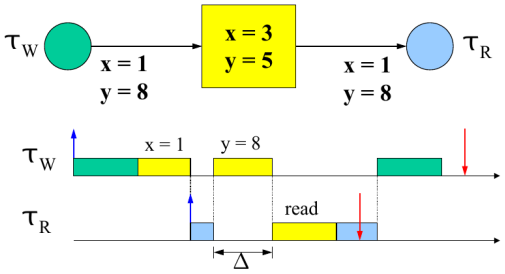


Figura 1.4: *mutual exclusion*

Mentre si analizza un *tasks set* e si cerca che il tempo di esecuzion sia vincolato da vincoli imposti in fase di progettazione, ad esempio $t_r \leq 10$, anche se si aumenta il numero di processori, si diminuisce il tempo di esecuzione dei task o si rilassano i vincoli

di precedenza, se non si uno ***scheduler*** appropriato si rischia in ogni caso di missare i vincoli imposti. L'apporccio più ***safe*** è quello di utilizzare meccanismi predicibili del kernel e analizzare il sistema per predirne il comportamento. La concorrenza deve essere progettata utilizzando:

- appropriati algoritmi di ***scheduler***.
- appropriati protocolli di **sincronizzazione**.
- efficienti meccanismi di **comunicazione**.
- predicibilità negli ***interrupt handling***.

Capitolo 2

Non Real-time scheduling algorithms

Capitolo 3

Real-time scheduling algorithms