SISTEMI IN TEMPO REALE

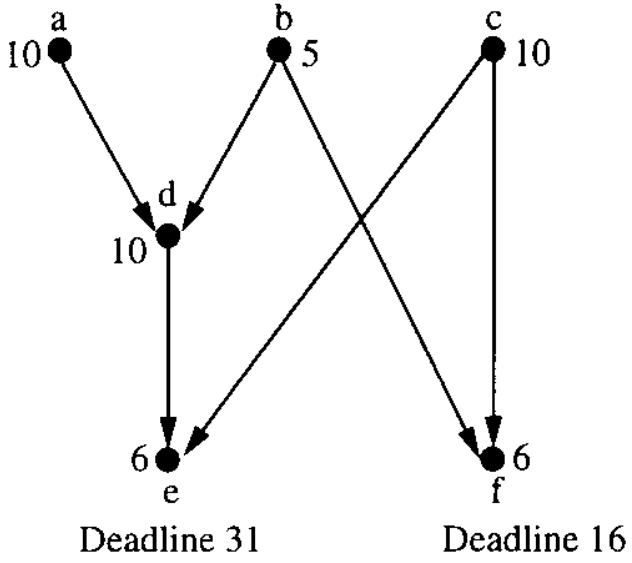
Strongly recommended readings:

Scheduling Algorithms and O.S. support for Real-Time Systems (January 1994)

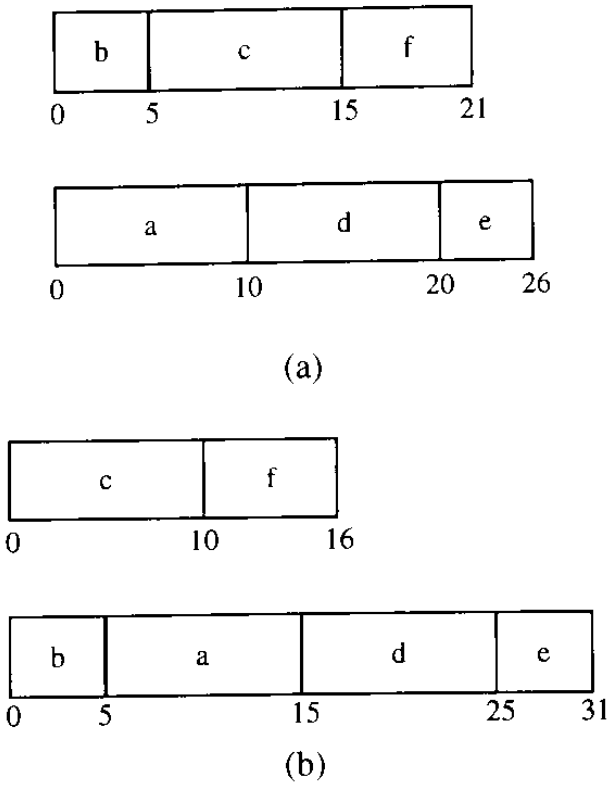
Real-Time Computing: A New Discipline of Computer Science and Engineering (January 1994)

Definizione [Stankovic 88]

I sistemi real-time (RT) sono quei sistemi di calcolo in cui la correttezza di funzionamento non dipende soltanto dalla validità dei risultati ottenuti, ma anche dal tempo in cui i risultati sono prodotti.



Example of a real-time application.



(a) Infeasible schedule; (b) Feasible schedule

Tratto da

appuntidigitali
il blog italiano sulla tecnologia

di **Alessio Di Domizio** - martedì 19 gennaio 2010

guest post di ***Emanuele Rampichini***

Troppo spesso nel mondo dell'informatica c'è confusione riguardo al concetto di "tempo reale". Siamo abituati ad utilizzare sistemi operativi che rispondono in maniera immediata ai nostri comandi, siano essi impartiti da una tastiera da un mouse o da altre periferiche. Questa abitudine porta a pensare che un sistema possa essere considerato in "tempo reale" semplicemente in relazione a quanto più rapidamente le risposte ai nostri stimoli vengano interpretate, processate e le risposte restituite.

Nei sistemi operativi interattivi tradizionali il tempo medio di risposta di un processo è il parametro su cui si gioca la partita. Nei sistemi real-time i concetti centrali sono invece quelli della garanzia e della prevedibilità. Non ci importa che un certo numero di processi vengano eseguiti complessivamente velocemente ma vogliamo delle garanzie temporali sui singoli processi (tecnicamente parliamo di ***deadline***).

Per rendere chiaro questo concetto possiamo fare un esempio molto semplice:

Supponiamo che alle 8:00 di mattina vi vengano assegnati i seguenti compiti:
- Pagare la bolletta
- Comprare il Pane

Supponiamo che per qualche motivo vi troviate di fronte a queste due possibilità:
- Passare immediatamente alle poste riuscendo a concludere il pagamento per le 10:00, poi al supermercato completando i compiti assegnati alle 11:50
- Passare prima al supermercato riuscendo a comprare il pane per le 9:30 e solo in seguito alle poste completando i compiti assegnati alle 11:00

Ovviamente nessuna persona (sana di mente) conoscendo le due alternative sceglierebbe la prima possibilità rispetto alla seconda.

Ma se cambiassimo le carte in tavola e inserissimo ulteriori vincoli nel problema?
- La bolletta va pagata entro le 10:30 pena il pagamento di una sovratassa
- Il pane va acquistato entro le 12:00 pena il salto del secondo pasto più importante della giornata

Ecco che quella che fino a un secondo fa ci sembrava una scelta illogica adesso diventa non solo comprensibile ma addirittura la scelta preferibile. Arrivare alle 11:50 e non alle 11:00 a casa non compromette in alcun modo il pranzo e non implicherà l'esborso di una sovratassa.
Ecco come il concetto di tempo reale esula da quello di velocità assoluta di risposta ed entra nel contesto dell'ambiente in cui si opera (nell'esempio l'ambiente sono i vincoli inseriti in un secondo momento).

Svincolandosi dall'esempio descritto sopra e dalla panoramica generale, problemi che vanno affrontati nell'ottica del "tempo reale" sono ovviamente ampiamente presenti nella società moderna. Basti pensare ai sistemi ABS delle automobili, o ai sistemi di controllo e stabilizzazione degli aeroplani.

SISTEMI IN TEMPO REALE

L'interesse per i sistemi RT è motivato dalle numerose applicazioni che richiedono questo tipo di elaborazione, tra le quali:
- la regolazione di impianti chimici e nucleari;
- il controllo di processi produttivi complessi;
- la robotica;
- i sistemi di controllo di volo sugli aerei;
- i sistemi di monitoraggio ambientale;
- i sistemi di controllo del traffico;
- i sistemi di telecomunicazione;
- i sistemi di regolazione delle automobili;
- i sistemi militari;
- le missioni spaziali.

Nonostante la vastità dei settori applicativi interessati, la maggior parte delle applicazioni RT ***viene ancora progettata con tecniche empiriche***:

- uso di processori sufficientemente veloci;
- grosse porzioni di codice assembler;
- driver di basso livello per la gestione di I/O;
- manipolazione delle priorità dei processi.

SISTEMI IN TEMPO REALE

Svantaggi delle tecniche empiriche

- **Programmazione laboriosa**: lunghi tempi di sviluppo e affidabilità dipendente dal programmatore.
- **Difficile comprensibilità del codice**: l'ottimizzazione peggiora la comprensibilità.
- **Scarsa manutenibilità del software**. La modifica di grossi programmi assembler risulta di estrema difficoltà anche per lo stesso progettista.
- **Difficile verifica dei vincoli temporali**. Senza l'ausilio di tecniche specifiche, risulta praticamente impossibile garantire il rispetto dei vincoli temporali.

La conseguenza di tutto ciò è una

==> **elevata inaffidabilità**

Esempi di incidenti causati da errori software:
- il primo volo dello Space Shuttle fu ritardato a causa di un errore di temporizzazione su uno dei processori di controllo;
- il sistema di difesa dei missili Patriot fallì durante la Guerra del Golfo per un errore di arrotondamento sul real-time clock.

SISTEMI IN TEMPO REALE

Come si possono evitare questi errori?

- Il testing non è una soluzione sufficientemente sicura. I processori dello Shuttle e dei Patriot erano stati accuratamente testati e nessun errore rilevato.
- Nelle applicazioni RT, il flusso dei programmi dipende dai dati di ingresso, i quali non possono essere riprodotti come nelle reali condizioni operative.

Soluzioni più sicure

- ***metodologie di progetto*** più accurate;
- tecniche di ***analisi statica*** del codice;
- ***meccanismi di nucleo*** adatti alla gestione di processi con vincoli temporali.

Un sistema RT deve essere sempre progettato per resistere alla peggiore combinazione possibile di eventi.

PERCHÈ

LE LEGGI DI MURPHY

LEGGE GENERALE
Se qualcosa può andar male, lo farà.

COSTANTE DI MURPHY
Le cose vengono danneggiate in proporzione al loro valore.

LEGGE DI NAESER
Si può fare qualcosa a prova di bomba, non di iella.

LEGGE DI GREEN
Se un sistema è stato concepito per essere tollerante ad un insieme di guasti, ci sarà sempre un idiota abbastanza ingegnoso da causare un guasto non tollerato.
Corollario
I cretini sono sempre più ingegnosi delle precauzioni che si prendono per impedirgli di nuocere.

SECONDA LEGGE DI SODD
Prima o poi, la peggiore combinazione possibile di circostanze è destinata a verificarsi.

SISTEMI IN TEMPO REALE

Cosa significa tempo reale?

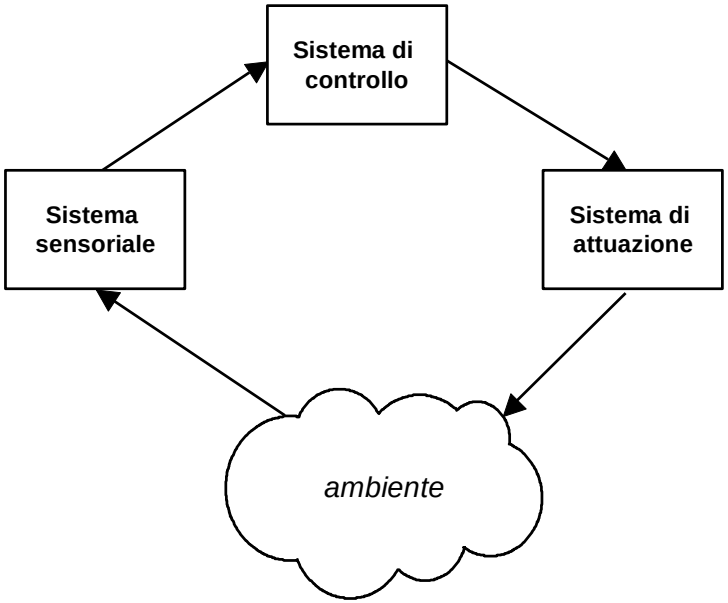
Nell'espressione *tempo reale* sono sintetizzati due concetti fondamentali:

TEMPO: la validità dei risultati di un processo di calcolo dipende anche dal tempo entro cui essi sono prodotti.
REALE: il tempo di sistema deve essere uguale a quello dell'ambiente in cui il sistema opera.

REAL-TIME non significa VELOCE.

Il termine *"veloce"* ha un significato relativo che ha poco senso se non è riferito all'ambiente in cui il sistema opera.

Dunque, le caratteristiche RT di un sistema sono strettamente legate all'ambiente in cui il sistema deve operare.



SISTEMI IN TEMPO REALE

Obiezione

Non vale la pena di studiare i sistemi RT poiché i requisiti temporali possono essere soddisfatti da un'architettura sufficientemente veloce.

Qualsiasi sia la velocità di un calcolatore, occorre sempre dimostrare che i vincoli temporali di un'applicazione sono garantiti.

REAL-TIME ≠ VELOCE

- L'obiettivo di una elaborazione veloce è di minimizzare il tempo medio di risposta di un insieme di processi.
- L'obiettivo di una elaborazione real-time è di soddisfare i requisiti temporali individuali di ciascun processo.

Quando i processi hanno dei requisiti individuali che devono essere rispettati, le prestazioni medie assumono scarsa rilevanza.

". . . ci fu un uomo che annegò attraversando un torrente con una profondità media di 15 centimetri!".

La proprietà più importante che un sistema real-time deve possedere non è la velocità, ma la **predicidibilità** .

PREDICIBILITÀ DEI SISTEMI IN TEMPO REALE

Predicibilità di un sistema

Per predicibilità di un sistema si intende la capacità di determinare in anticipo se i processi di calcolo potranno essere completati entro i vincoli temporali desiderati.

La predicibilità è una misura a priori del livello di rispetto dei vincoli temporali.

La predicibilità è misurata dalla percentuale di processi per cui è garantito il rispetto dei vincoli.

Il tipo di predicibilità può essere **deterministica** (*completamente predicibile*), **probabilistica** (*una frazione dei processi è garantita o la probabilità che un qualunque task sia garantito non è inferiore ad una certo valore*) oppure **deterministica a run-time** (*all'atto dell'attivazione del processo è possibile determinare se esso può essere garantito*).

DIPENDENZA DELLA PREDICIBILITÀ DAL S.O.

La predicibilità dipende da diversi fattori, che vanno dalle caratteristiche architetturali della macchina fisica, ai meccanismi di nucleo, fino al linguaggio di programmazione.

- Le tecniche di DMA;
- La cache;
- Il meccanismo delle interruzioni;
- Le primitive del nucleo;
- Il meccanismo semaforico;
- La gestione della memoria;
- Il linguaggio di programmazione.

DIPENDENZA DELLA PREDICIBILITÀ DAL S.O.

Ci sono diversi fattori che causano la *non prevedibilità* nella risposta del sistema operativo. Tra di essi, i principali sono i seguenti:

- Il **DMA**: ***può rubare il bus alla CPU ritardando l'esecuzione di un task critico***. In un sistema real-time si preferisce quindi disattivarlo o usarlo in modalità *time-slice* dove si assegna in maniera costante e fissa il bus al DMA anche se non ci sono operazioni da fare.
- La **cache**: ***può causare non prevedibilità poiché esistono casi in cui essa fallisce e può causare ritardi nell'accesso alla memoria da parte della CPU***. Dovendo considerare quindi il caso peggiore si preferisce non usarla affatto.
- **Meccanismi di gestione della memoria**: ***queste tecniche non devono introdurre ritardi non prevedibili durante l'esecuzione di task critici***; ad esempio la paginazione può causare dei *page fault* intollerabili per un sistema hard real-time. Tipicamente si usa la segmentazione o la partizione statica della memoria.
- Le **interruzioni**: sono generate da dispositivi periferici quando hanno qualche informazione da scambiare con la CPU. Queste interruzioni ***durante l'esecuzione di un task critico generano ritardi non prevedibili*** e quindi si preferisce disattivarle.
- I **sistemi di power management**: sono ***meccanismi hardware che possono rallentare la CPU o far eseguire ad essa del codice utile a dissipare minor energia***. È chiaro che in un sistema real-time è importante non sfondare una deadline piuttosto che consumare poca energia, quindi questi meccanismi vengono disattivati.

SCELTA DEL SISTEMA OPERATIVO REAL TIME

Tra i Real Time OS (RTOS) commerciali troviamo sistemi operativi POSIX-conformant (ad esempio **LynxOS** che è Unix compatibile) e non POSIX-conformant come ad esempio **VxWorks** (che supporta in parte gli standard POSIX). Per quanto riguarda i sistemi Open Source è possibile l'uso di **Linux**, con opportune precauzioni, o di **RTAI**/**Xenomai**.

Problemi del real-time in Linux

- Lo ***schedulatore***, che ha come obiettivo l'assegnazione della CPU ai vari processi, non può conoscere i requisiti temporali dei vari processi real-time.
- Anche se si usa la schedulazione FIFO o FIFO Round Robin, che tendono ad aumentare la prevedibilità del sistema, non si hanno comunque garanzie sui ritardi introdotti dalle chiamate di sistema e dalle attività del kernel.

Esempi di sistemi operativi RT

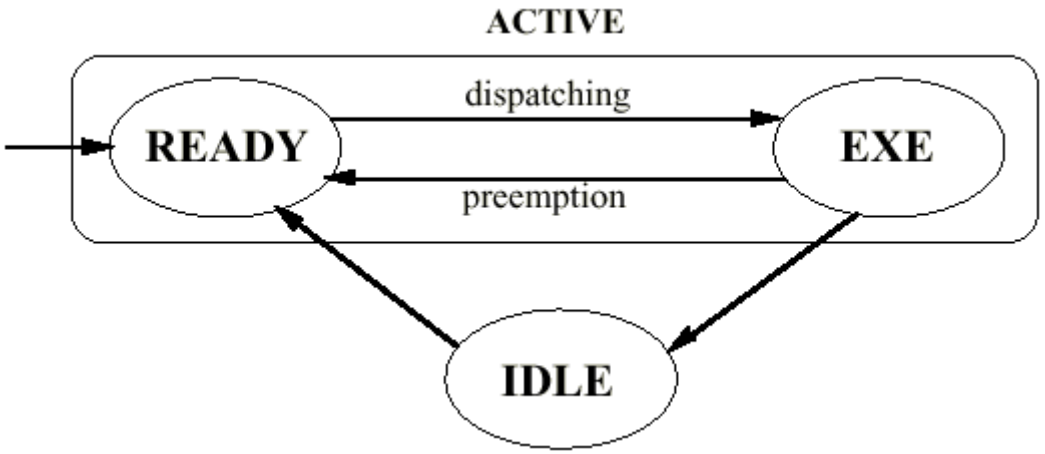
Per conoscere i sistemi operativi, sia Open Source che proprietari, in grado di lavorare in real-time su adeguate architetture hardware, si rimanda a quanto riportato su Wikipedia.

SISTEMI IN TEMPO REALE E PROCESSI

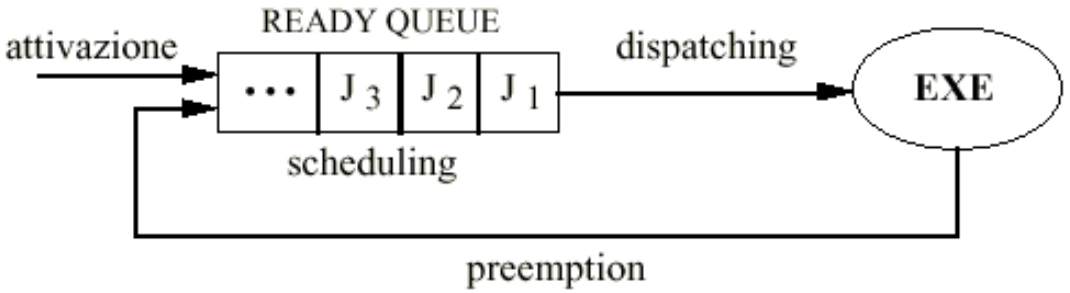
Ciclo di esecuzione di un processo

Processo (o task): **è una sequenza di istruzioni che, in assenza di altre attività, viene eseguita dal processore in modo continuativo fino al suo completamento**.

Su un singolo calcolatore un processo può essere nei seguenti stati:



- **Algoritmo di scheduling**: **insieme di regole che determinano la scelta del processo da mandare in esecuzione tra i processi pronti.**
- **Dispatching: assegnazione effettiva del processore al processo pronto schedulato.**



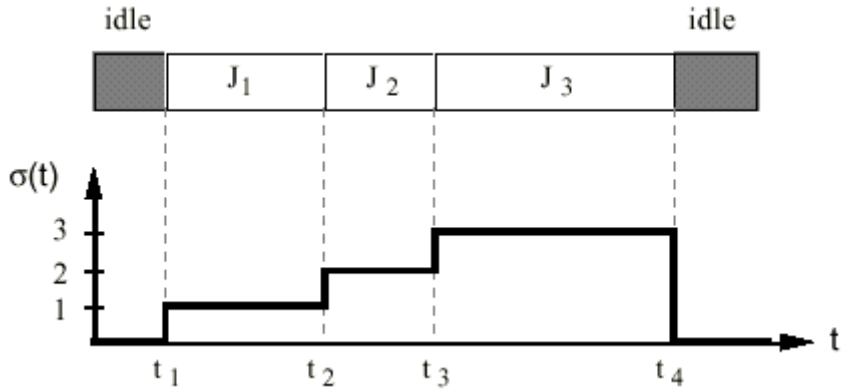
SCHEDULING DI SISTEMI IN TEMPO REALE

Schedulazione

Dato un insieme di task $\boldsymbol{J} = \{J_1, J_2, ..., J_n\}$, una schedulazione è **un assegnamento dei task al processore**.
E' una funzione σ**:** $\mathbf{R}^+ \rightarrow \mathbf{N}$ tale che:

$\forall\ t \in \mathbf{R}^+$, $\exists\ t_1, t_2$ tali che $t \in [t_1, t_2)$ e inoltre
$\forall\ t' \in [t_1, t_2)$ si abbia $\sigma(t) = \sigma(t')$.

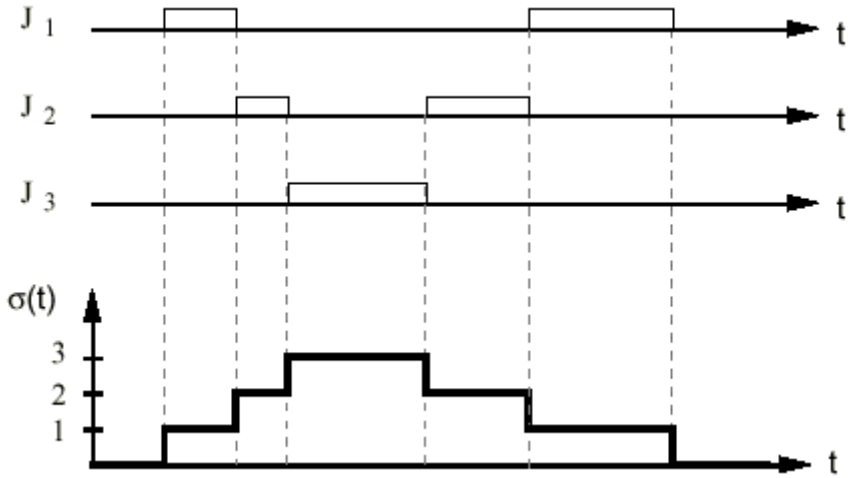
$\sigma(t) = k$ con $k \neq 0$, indica che all'istante t il processore esegue il task J_k.
$\sigma(t) = 0$ indica che il processore è libero (**idle**).



- Negli istanti t_1, t_2, t_3, e t_4, si ha una commutazione di contesto (*context switch*).
- Ciascun intervallo $[t_i, t_{i+1})$ in cui $\sigma(t)$ è costante prende il nome di *time slice*.

PROPRIETA' DELLO SCHEDULING

- Una schedulazione si dice ***preemptive*** se il task in esecuzione può essere sospeso per assegnare il processore ad un altro task.



- Una schedulazione è ***fattibile*** se esiste un assegnamento dei task al processore tale che tutti i task siano completati rispettando un insieme di vincoli prefissati.

- Un insieme di task ***J*** si dice ***schedulabile*** se per esso esiste una schedulazione fattibile.

TIPI DI VINCOLI DEI SISTEMI IN TEMPO REALE

Vincoli sui processi

- vincoli ***temporali***;
- vincoli ***di precedenza***;
- vincoli ***su risorse condivise***.

Vincoli temporali

Un processo real-time è caratterizzato da una scadenza temporale prefissata, che prende il nome di ***deadline***.

Un risultato prodotto oltre la propria deadline non è solo in ritardo, ma è dannoso .

A seconda delle conseguenze provocate da una mancata deadline, i processi real-time vengono solitamente distinti in tre tipi: *hard*, *firm* e *soft*.

Hard real-time: se la violazione della deadline comporta un effetto catastrofico sul sistema.

Firm real-time: se la violazione della deadline non comporta effetti catastrofici, ma l'utilità dei risultati decresce con il crescere del ritardo rispetto alla scadenza.

Soft real-time: se la violazione della deadline non compromette il corretto funzionamento del sistema.

SISTEMA OPERATIVO E TEMPO REALE

Un sistema operativo per gestire processi real-time hard è detto **Hard Real-Time Operating System (HRT OS)**.

Tipi di processi in tempo reale

Tipici processi hard:
- acquisizione di dati sensoriali;
- rilevamento di condizioni critiche;
- asservimento di attuatori;
- pianificazione di azioni senso-motorie;
- controllo di dispositivi automatici.

Tipici processi firm:
- transazioni di un data-base;
- in generale le applicazioni on-line.

Tipici processi soft:
- ingresso di caratteri da tastiera;
- interprete di comandi utente;
- visualizzazione di messaggi su monitor;
- rappresentazione dello stato del sistema.

SISTEMI IN TEMPO REALE

Vincoli di precedenza

Le relazioni di precedenza eventualmente esistenti tra i processi vengono espresse mediante un grafo diretto e aciclico.

Vincoli su risorse

Una risorsa è una entità software che può essere utilizzata da uno o più processi per portare a termine la propria esecuzione.

Una risorsa può essere una struttura dati, una zona di memoria, o una porzione di codice.

Una risorsa può essere:

- ***condivisa*** se utilizzata da più processi;
- ***dedicata*** se è riservata ad un solo processo.

Una risorsa si dice mutuamente esclusiva se non consente l'accesso contemporaneo di più processi.

Una porzione di codice mutuamente esclusiva prende il nome di ***sezione critica***.

Tipicamente la mutua esclusione è realizzata mediante il meccanismo semaforico, gestito dalle primitive ***wait*** e ***signal*** (o *lock* e *unlock*).

Rispettare un vincolo su risorsa significa quindi consentire un uso della risorsa mutuamente esclusivo.

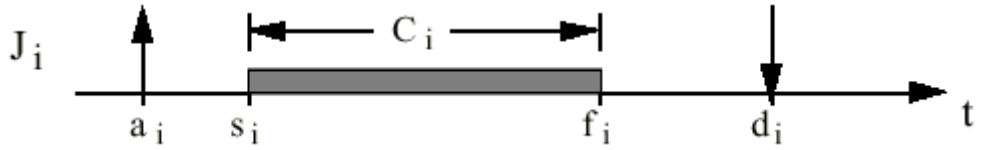
I processi in attesa di risorse condivise occupate si dicono bloccati sulla risorsa, e vengono accodati in una coda associata al meccanismo di protezione della risorsa.

SISTEMI IN TEMPO REALE

Caratteristiche dei processi in tempo reale

ready time **a**: tempo in cui il task diventa pronto per l'esecuzione;
computation time **C**: tempo necessario al processore per eseguire completamente il task senza interruzioni;
deadline **d**: tempo entro cui il task deve essere completato;
start time **s**: tempo in cui il task va in esecuzione per la prima volta;
finishing time **f**: tempo in cui il task termina la sua esecuzione;
criticalness r: criticità della deadline (hard/firm/soft);
value **v**: importanza relativa del task all'interno di una classe di criticità.

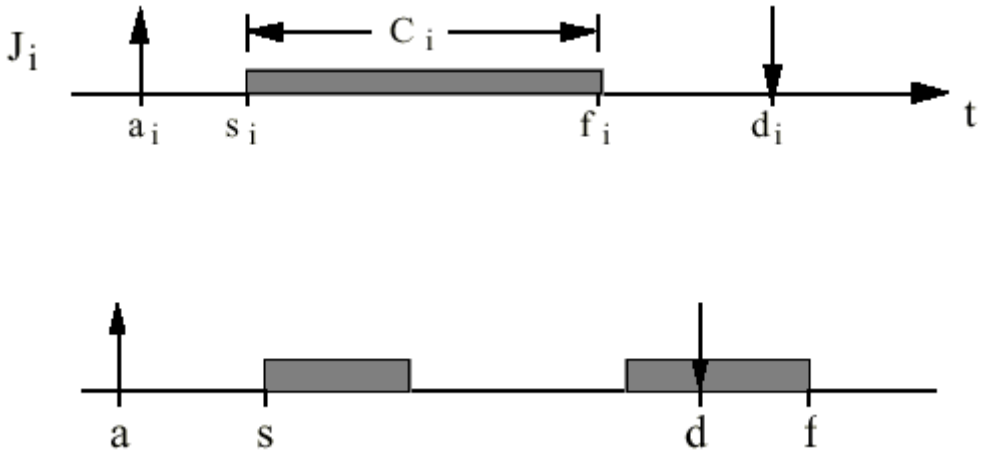
Si tenga conto che il tempo in cui viene rilevata la necessità di attivare il task è antecedente al tempo **a**.



SISTEMI IN TEMPO REALE

Variabili temporali dei processi

lateness **L**: $L = f - d$. E' il ritardo di completamento del task rispetto alla deadline.
tardiness **E**: $E = max(0, L)$. E' detto anche exceeding time ed è il tempo in cui il task rimane attivo oltre la deadline.
laxity **LX**: $LX = d - a - C$. E' detto anche *slack time* ed è il ritardo di attivazione massimo che un task può subire per non eccedere la sua deadline.



PROCESSI PERIODICI E APERIODICI

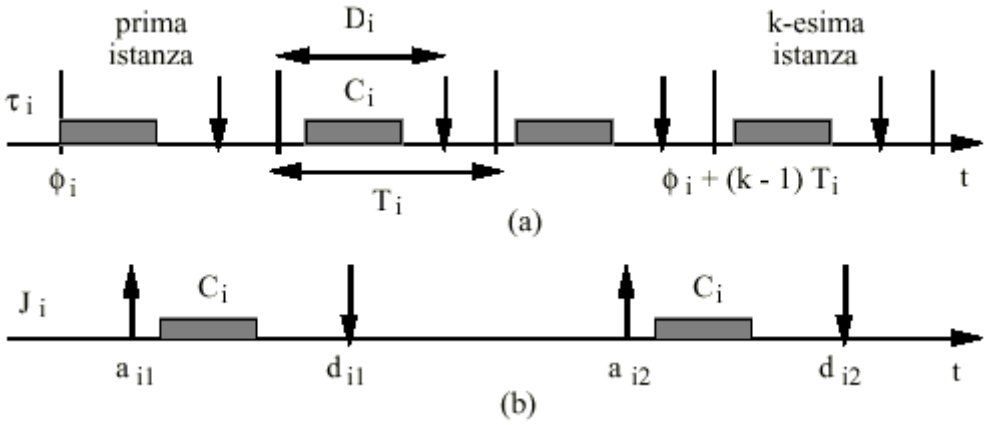
Processi periodici: consistono in una sequenza infinita di attività identiche, ciascuna delle quali prende il nome di ***istanza*** ($\tau_{i,k}$).

La regolarità deve essere garantita dal sistema (time driven):

$$r_{i,k} = \phi_i + (k\text{-}1)\, T_i$$

$$d_{i,k} = r_{i,k} + D_i$$

Processi aperiodici: consistono in una sequenza di attività identiche J_i, ciascuna delle quali caratterizzata da un tempo di arrivo (event driven), un tempo di calcolo, e una deadline.



SCHEDULING DEI PROCESSI

Problema generale di scheduling

Dato un insieme di *n* task $\boldsymbol{J} = \{J_1, J_2, ..., J_n\}$ di cui sia noto il grafo dei vincoli di precedenza e le modalità di utilizzo delle risorse condivise, il problema dello scheduling consiste nel trovare una schedulazione fattibile dell'insieme ***J***, ossia una schedulazione tale che:

- tutti i task terminino entro la propria deadline;
- tutti i vincoli di precedenza siano rispettati;
- tutte le risorse condivise mutuamente esclusive siano utilizzate correttamente.

Garey e Johnson [Gar 75] hanno dimostrato che tale problema appartiene alla classe **NP hard**. Ossia non può essere risolto in tempo polinomiale rispetto al numero di processi.

Per trattare il problema in tempo polinomiale, si fanno delle ipotesi semplificative sul sistema e/o sui vincoli tra i processi.

Tipiche ipotesi semplificative

- singolo processore
- preemption
- attivazioni simultanee
- assenza di vincoli di precedenza
- assenza di vincoli su risorse

ALGORITMI DI SCHEDULING

Classificazione di algoritmi

***Preemptive*:** ammettono preemption in qualsiasi istante.
***Non preemptive*:** ricevuto il processore, il task non può essere interrotto.

***Statici*:** le decisioni sono basate su parametri fissi assegnati ai processi prima della loro attivazione.
***Dinamici*:** le decisioni sono basata su parametri che possono variare durante l'esecuzione dei processi.

***Off-line*:** tutte le decisioni di scheduling sono prese prima dell'attivazione dei processi, su dati noti a priori.
***On-line*:** le decisioni di scheduling vengono prese a run-time sull'insieme dei processi attivi.

***Best-Effort*:** privilegiano le prestazioni medie.
***Guaranteed*:** garantiscono il rispetto dei vincoli temporali di ogni singolo task.

PARADIGMI DI SCHEDULING

Algoritmi statici table-driven

Gli approcci di questo tipo si basano su analisi off-line di schedulabilità, i cui risultati danno origine a "tavole di schedulazione" da usare a run-time per decidere quando un task deve iniziare la sua esecuzione. Tali algoritmi cercano di soddisfare i requisiti di applicazioni critiche, che richiedono la garanzia a priori delle deadline e quindi addirittura la pre-allocazione dei task. Questi ultimi sono ovviamente di natura periodica.
Alcuni algoritmi alla base di questi approcci: *Least Common Multiple* (LCM) e *Earliest Deadline First* (EDF)

Algoritmi statici priority-driven, con applicazione della preemption

Anche questi approcci si basano su analisi off-line. Ai vari task sono assegnate priorità che possono essere applicate staticamente o dinamicamente, consentendo allo scheduler di requisire la CPU da un task a più bassa priorità, per assegnarla ad uno a più alta priorità.
Alcuni algoritmi alla base di questi approcci: *Rate Monotonic* (RM) e *Least Laxity First* (LLF)

Algoritmi dinamici planning-based

Questo approccio, dinamico, tende a verificare, quando insorge la necessità di eseguire un nuovo task, la fattibilità di uno scheduling che includa il task. Se non viene individuato uno scheduling fattibile, il task non viene accettato. Viene a volte eseguito quando la deadline è sufficientemente lontana ed in caso di esito negativo dell'analisi di fattibilità sia possibile ricorrere ad un'azione alternativa.
Approcci distribuiti (in presenza di più processori) sono quelli del *focussed addressing* (il task viene inviato al processore stimato avere il più alto surplus di risorse e tempo di CPU) e del *bidding* (il task viene inviato al processore che ha garantito, rispondendo ad una sorta di asta, il minor tempo di completamento del task stesso).

Algoritmi dinamici best effort

Si basano sull'attribuzione a ciascun task di una priorità basata sui suoi attributi e su approcci che combinano in maniera empirica algoritmi EDF, LLF o di puro rispetto della priorità.