Introduzione

RTS

Anno accademico 2008/9
Sistemi *Real-Time*

Tullio Vardanega, tullio.vardanega@math.unipd.it

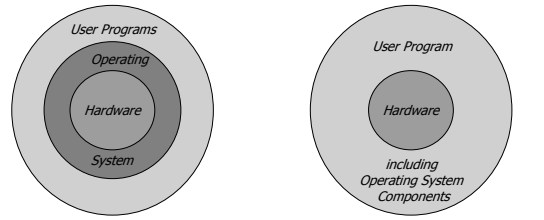
Una prima definizione informale – 1

- **Sistema *real-time***
  - **Una combinazione di computer, dispositivi di I/O e *software* specializzato, caratterizzata da**
    - Forte interazione con l'ambiente esterno
    - Mutevolezza nel tempo dello stato dell'ambiente esterno
    - Il sistema deve simultaneamente controllare diverse parti dell'ambiente e reagire a suoi cambiamenti
  - **Le attività del sistema sono naturalmente concorrenti**
  - **Le attività del sistema sono soggetti a vincoli temporali**
    - Di reattività, accuratezza, durata, completamento
  - **Il soddisfacimento dei vincoli deve essere dimostrabile**

Una prima definizione informale – 2

- **Sistema *real-time***
  - **La sua correttezza dipende non solo dal risultato logico prodotto ma anche dal tempo in cui ciò avviene**
    - La risposta ha una funzione di utilità che dipende dall'applicazione
    - Correttezza logica e correttezza temporale
  - **Rispondere fuori tempo può essere tanto grave quanto produrre la risposta sbagliata**
- **Sistema *embedded***
  - **Il *computer* e il suo *software* sono immersi in un sistema ingegneristico che ne costituisce l'ambiente**

Sistema *embedded*



Typical General-Purpose Computing Configuration ***Typical Embedded Computing Configuration***

Caratteristiche salienti – 1

- **Complessità**
  - **Sia algoritmica che di verifica**
- **Eterogeneità delle componenti e delle attività**
  - **Per discipline scientifiche e per interazione con l'ambiente**
- **Dimensione estremamente variabile**
  - **Da minuscolo con scarse risorse a enorme ma con risorse finite**
- **Garanzia di affidabilità e sicurezza**
  - ***Safety* più che *security***

Caratteristiche salienti – 2

- **Necessità di rispondere sia a eventi causati dall'ambiente ma anche al trascorrere del tempo**
  - **Natura *event-driven* a *clock-driven***
- **Continuità di operazione**
  - **Non deve dipendere dall'operatore umano**
- **Architettura *software* inerentemente concorrente**
- **Predicibilità temporale**
  - **Poter valutare staticamente (*off line*) l'adeguatezza del comportamento temporale**

False credenze – 1

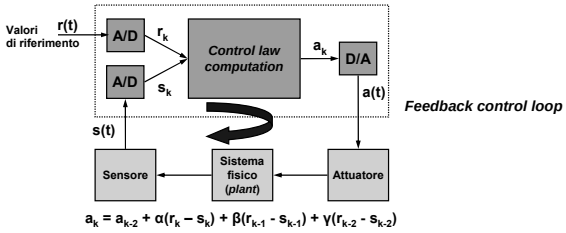
- La progettazione di sistemi *real-time* è empirica e non scientifica
  - Falso: lo vedremo qui
- La velocità dell'*hardware* consentirà di soddisfare qualunque esigenza temporale
  - Falso: lo sperimentiamo ogni giorno
- La computazione *real-time* è veloce
  - Falso: ci interessa la predicibilità
- La disciplina del *real-time* altro non è che *performance engineering*
  - Falso: vedremo qui di cosa è fatta

False credenze – 2

- La programmazione *real-time* è a basso livello
  - Falso: la verifica è tanto piu agevole quanto piu alto è il livello di programmazione
- I problemi *real-time* sono gia stati tutti risolti in altri ambiti dell'informatica
  - Falso: la ricerca operativa risolve problemi con tecniche probabilistiche e/o *one-shot*
  - Falso: l'informatica generale si interessa prevalentemente all'ottimizzazione prestazione del caso medio

Esempi – 1

- Sistemi digitali di sensori e attuatori

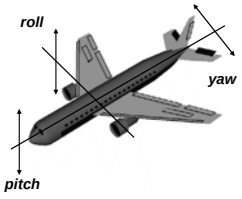


$a_k = a_{k-2} + \alpha(r_k - s_k) + \beta(r_{k-1} - s_{k-1}) + \gamma(r_{k-2} - s_{k-2})$

Esempi – 2

- Fattori di influenza
  - Qualità di risposta (*responsiveness*) percepita
    - Il campionamento dal sensore è periodico
    - Il comando all'attuatore viene prodotto al tempo del successivo campionamento
    - La qualità degrada con l'ampiezza del periodo
  - Capacità di risposta del sistema (*plant*)
    - Il miglior controllo ne limita l'oscillazione
    - Un sistema capace o bisognoso di reagire velocemente a variazioni nell'ambiente (***rise time*** R) richiede alta frequenza di attuazione e dunque veloce campionamento (periodo T)
    - Una buona proporzione R/T è nell'intervallo [10 .. 20]

Esempi – 3



- Sistemi complessi multi-periodo
  - Conviene fissare una relazione armonica tra i periodi T_i
    - Ma ciò causa indesiderabile accoppiamento tra azioni non correlate
  - Diverse componenti di velocità
    - Per esempio: *forward, side slip, altitude*
  - Diverse componenti di rotazione
    - Per esempio: *Roll, pitch, yaw*
  - Svariate attività (incluso il *control loop*) svolte con la loro frequenza

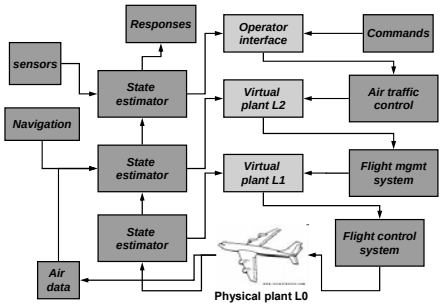
Esempi – 4

- Controllo di volo (*multi-rate* armonico)
  - Con ciclo 180 Hz
    - Convalida i dati di sensore e seleziona le sorgenti da campionare
      - In caso di errore riconfigura il sistema
    - Con ciclo 30 Hz (ogni 6 attivazioni)
      - Rileva ingressi da tastiera e selezione di modo operativo
      - Normalizza i dati e trasforma le coordinate
      - Aggiorna il riferimento
    - Con ciclo 30 Hz (ogni 6 attivazioni)
      - *Control law* per *pitch, roll, yaw* (*loop* esterno)
      - Integra i calcoli
    - Con ciclo 90 Hz (ogni 2 attivazioni)
      - *Control law* per *pitch, roll* (*loop* interno)
    - *Control law yaw* (*loop* interno) sulle uscite del calcolo a 90 Hz
    - Comanda gli attuatori
    - Esegui *test* di sanità

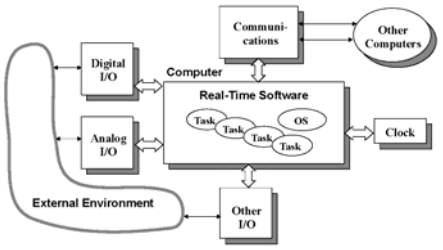
Esempi – 5

- **I sistemi di comando e controllo sono spesso organizzati gerarchicamente**
  - **Al livello più basso stanno i sistemi digitali che controllano parte del sistema fisico**
  - **Al livello più alto sta l'interfaccia con l'operatore umano**
    - L'uscita di un controllore di alto livello diventa valore di riferimento r(t) di qualche controllore di basso livello
  - **Più articolata la gerarchia più complessa l'interdipendenza logica e temporale**

Esempi – 6



Visione d'insieme



Una prima classificazione

- **Punto di vista tradizionale secondo gli algoritmi di controllo**
  - **Sistemi strettamente periodici**
    - Periodi armonici (forzati)
    - *Polling* per eventi non periodici
  - **Sistemi prevalentemente periodici**
    - Minor accoppiamento
    - Migliore risposta a eventi non periodici
  - **Sistemi prevalentemente non periodici ma predicibili**
    - Variazioni anche significative nell'arrivo degli eventi ma entro intervalli massimi fissati
  - **Sistemi non periodici e non predicibili**
    - *Another ballgame*

Definizioni – 1

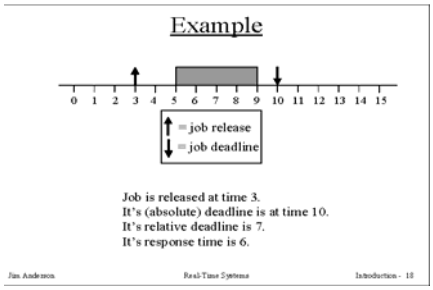
- ***Job***
  - **L'unità di lavoro mandata in esecuzione dallo *scheduler***
  - **Richiede risorse per eseguire**
- ***Task***
  - **Unità di composizione funzionale e architetturale**
  - **Un *job* è l'esecuzione dell'istanza di un *task***
- ***Release time***
  - **Quando un *job* dovrebbe diventare pronto per l'esecuzione**
    - Il corrispondente stimolo viene chiamato [*release*] *event*
    - Vi può essere ritardo tra l'arrivo del release event e quando il corrispondente *job* diventa *ready*
  - **Può essere a un distanza fissata (*offset*) dall'avvio del sistema**
    - L'*offset* del del primo *job* di un task T viene detto *phase* ed è un attributo di T

Definizioni – 2

- ***Deadline***
  - **Il tempo entro il quale l'esecuzione di un *job* deve completare**
    - Per esempio entro il successivo *release time*
  - **In generale è un valore arbitrario e può essere <, =, > del successivo *release time***
- ***Response time***
  - **L'estensione temporale di un *job* tra *release time* ed effettivo completamento**
  - **Il massimo *response time* ammissibile per un *job* viene detto *relative deadline***
  - **Si dice invece *absolute deadline* di *job* la somma algebrica tra *release time* e *relative deadline***

Introduzione

Esempio



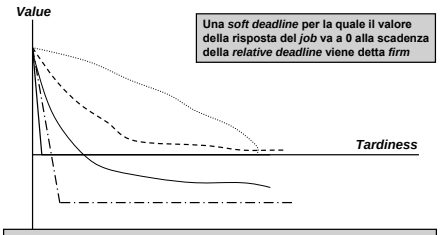
Introduzione

Definizioni – 3

- ***Hard deadline***
  - **Se le conseguenze di un completamento di *job* oltre la *deadline* assegnata siano assolutamente indesiderabili**
    - Il soddisfacimento richiede validazione
- ***Soft deadline***
  - **Conseguenze fastidiose ma tollerabili se occasionali**
    - La nozione di "occasionale" può essere fissata in termini probabilistica (X% delle volte) oppure come funzione di utilità
- ***Tardiness***
  - **La distanza temporale per un *job* tra *response time* e *deadline***
    - Vale 0 per un ogni completamento in tempo utile
- ***Usefulness***
  - **Valore di utilità del prodotto del *job* come funzione della sua *tardiness***
  - **Viene associata alla nozione di *laxity***

Introduzione

Funzione di utilità



Nozione interessante ma di applicazione tutt'altro che ovvia sia per definizione che per verifica

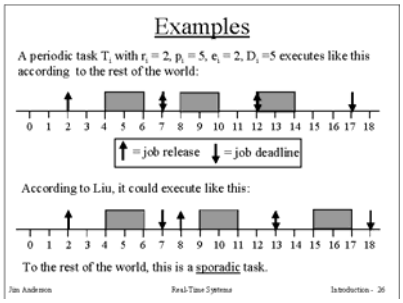
Introduzione

Modelli astratti – 1

- **Risorse**
  - **Attive (*processor*, *server*)**
  - **Passive (memoria, dati condivisi, semafori, ...)**
    - Possono essere riusabili se il loro uso non le consuma
    - Se sono sempre in quantità sufficiente per soddisfare le necessità (sono *plentiful*) possono essere escluse dallo spazio del problema
- **Parametri temporali**
  - ***[Release-time] Jitter***
    - Specifica la possibile variabilità del *release time*
  - ***Inter-arrival time***
    - Tempo di separazione tra *release time* di *job* successivi non strettamente periodici
      - ***Sporadic job*** se esiste un valore minimo garantito
      - ***Aperiodic job*** altrimenti

Introduzione

Task periodico e *task* sporadico



Introduzione

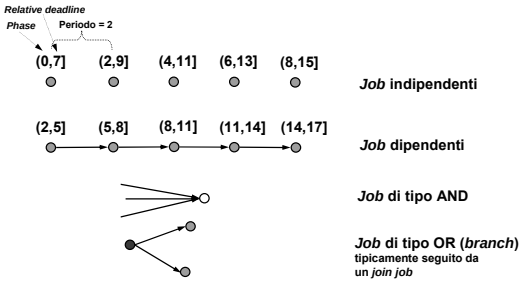
Modelli astratti – 2

- **Tempo d'esecuzione**
  - **Può non essere costante e variare tra un *best-case execution time* (BCET) e un *worst-case execution time* (WCET)**
- **Modello periodico**
  - **Composto di *job* periodici e sporadici**
  - **Accuratezza di rappresentazione decresce al crescere di *jitter* e variabilità del tempo d'esecuzione**
  - ***Hyperperiod* H dei Task $\{Ti\}_{i=1,..,N}$**
    - LCM (least common multiple) dei periodi {Pi}
  - ***Utilization***
    - Per ogni task Ti : rapporto tra il tempo d'esecuzione e il periodo Ui = Ei / Pi
    - Per il sistema (***total utilization***) : ∑i Ui

Modelli astratti – 3

- **Fissare i parametri d'esecuzione**
  - **Il tempo che intercorre tra quando un *job* periodico diventa *ready* e il successivo periodo P è certamente sempre minore di P**
  - **Fissare *phase* > 0 e deadline D < P per un *job* può essere utile per limitare il *jitter* del suo *response time***
  - **I *job* di un sistema possono essere tra loro indipendenti**
    - Possono eseguire in qualsiasi ordine
  - **Oppure possono essere soggetti a *precedence constraints***
    - Come tipicamente nel modello collaborativo *producer – consumer*

Grafi di precedenza estesi (*task graph*)



Modelli astratti – 4

- **Fissare i parametri funzionali**
  - **Ammissibilità del prerilascio di *job***
    - Può dipendere dal supporto a tempo d'esecuzione
    - Ma anche dallo stile di programmazione (*non-reentrancy*)
    - Il prerilascio ha onere temporale non nullo (ne parleremo)
  - **Importanza (*criticality*) di *job***
    - Si può tradurre in una priorità d'esecuzione
    - In generale indica quali attività vanno garantite anche eventualmente a scapito di altre
  - **Ammissibilità del prerilascio delle risorse**
    - Qualche risorsa è intrinsecamente prerilasciabile (per fortuna! quale?)
    - Altre non ammettono prerilascio (una delle 4 precondizioni del *deadlock*)

Modelli astratti – 5

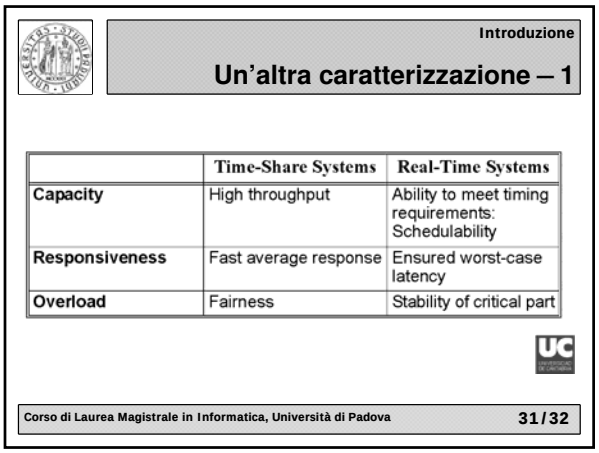
- **Selezione di *job* per l'esecuzione**
  - **Lo *scheduler* assegna *job* a risorse *processor***
  - **L'assegnamento risultante viene detto *schedule***
  - **Uno *schedule* è valido se**
    1. Ogni *processor* è assegnato a non più di un *job* alla volta
    2. Ogni *job* è assegnato a non più di un *processor* alla volta
    3. Nessun *job* è selezionato per l'esecuzione (*scheduled*) prima del suo *release time*
    4. In relazione agli algoritmi di *scheduling* la quantità di *processor time* assegnato a un *job* non è inferiore sul suo BCET e non superiore al suo WCET
       - Ricorda: un algoritmo di *scheduling* attua una politica
    5. Tutti i vincoli di precedenza in vigore tra *task* e tra risorse sono soddisfatti

Modelli astratti – 6

- **Uno *schedule* valido è detto *feasible* (meglio: corretto)**
  - **Se i vincoli temporali di ogni *job* in esso sono soddisfatti**
- **Un insieme di *job* è detto *schedulable* da un algoritmo di *scheduling***
  - **Se quell'algoritmo produce sempre uno *schedule* valido per quel problema**
- **Un algoritmo di *scheduling* è *optimal***
  - **Se produce sempre uno schedule *feasible* ove ne esista almeno uno**
- **In un sistema reale vi sono diversi *scheduler* che operano in gerarchia**
  - **Qualcuno governa l'accesso a risorse logiche (passive)**
  - **Altri l'accesso a risorse fisiche (attive)**

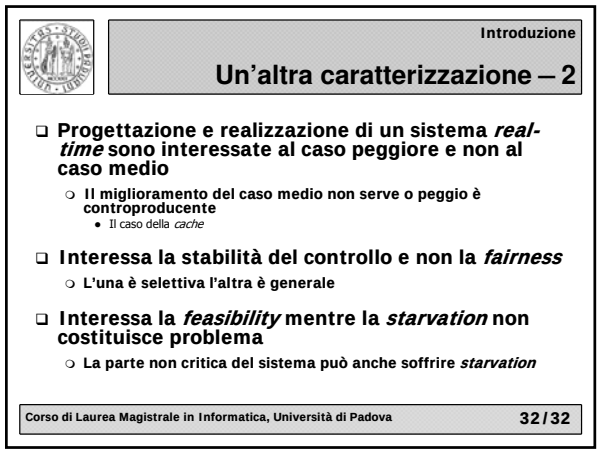
Modelli astratti – 7

- **2 algoritmi sono dunque di primo interesse per i sistemi *real-time***
  - **Quello di *scheduling* che vorremmo fosse *optimal***
    - Problema relativamente semplice
  - **Quello di *feasibility analysis***
    - Problema decisamente più complesso
- **Per la comunità scientifica ma non sempre consistentemente**
  - **I *test* di *feasibility* sono esatti**
    - Provano *True* o *False* (sono necessari e sufficienti)
  - **I *test* di *schedulability* sono solo sufficienti**



Un'altra caratterizzazione – 1

	Time-Share Systems	Real-Time Systems
Capacity	High throughput	Ability to meet timing requirements: Schedulability
Responsiveness	Fast average response	Ensured worst-case latency
Overload	Fairness	Stability of critical part



Un'altra caratterizzazione – 2

- **Progettazione e realizzazione di un sistema *real-time* sono interessate al caso peggiore e non al caso medio**
  - **Il miglioramento del caso medio non serve o peggio è controproducente**
    - Il caso della *cache*
- **Interessa la stabilità del controllo e non la *fairness***
  - **L'una è selettiva l'altra è generale**
- **Interessa la *feasibility* mentre la *starvation* non costituisce problema**
  - **La parte non critica del sistema può anche soffrire *starvation***