Laurea Magistrale in Ingegneria Informatica
Laurea Magistrale in Ingegneria Elettronica e Telecomunicazioni per lo Sviluppo Sostenibile

Sistemi e Tecnologie per l'Automazione LM

Sistemi di elaborazione Real Time per l'Automazione

Ing. Gianluca Palli
DEI - Università di Bologna
Tel. 051-2093186
E-mail: gianluca.palli@unibo.it
http://www-lar.deis.unibo.it/people/gpalli/

Revisionato il 08/10/2015

Introduzione

- Le funzioni di controllo sono strettamente legate al tempo
- I sistemi di elaborazione delle informazioni usati per il controllo dovranno sempre rispettare specifiche temporali
- Nell'ambito elettronico/informatico i sistemi di elaborazione con vincoli temporali sono detti sistemi real time
  - Non sono usati solo nell'automazione
  - Telefonia… telecomunicazioni in genere

Obiettivo

- Introdurre i concetti generali relativi ai sistemi di elaborazione real time
  - Non solo per automazione
  - Per approfondimenti si rimanda a corsi specializzati
- Mappare le funzioni di controllo nell'ambito dell'elaborazione real time
- Introdurre soluzioni realizzative di sistemi real time
  - Enfasi su accorgimenti tipici per l'automazione

Sommario

- Sistemi Real Time: Scopo e Definizioni
- Applicazioni Real Time, Eventi e Classificazione
- Sistemi Real Time e Parallelismo delle Applicazioni
  - Cenni a Programmazione Concorrente nei Sistemi Real Time
  - Cenni ad Algoritmi di Scheduling per Sistemi Real Time
- Sistemi Real Time e Sistemi di Controllo
- Sistemi Operativi Real Time: Gestione delle Attività time-driven e event-driven

Sistemi di Elaborazione digitale Real Time: Definizioni

Sistemi Real Time: definizioni

- Un sistema per la manipolazione dell'informazione che debba fornire risultati/risposte entro certi tempi viene usualmente detto "Sistema Real-Time"
  - Due tipologie di correttezza devono essere garantite:
    - Correttezza logica:
      i risultati/risposte forniti devono essere quelli previsti (normalmente richiesta a qualunque sistema di elaborazione)
    - Correttezza temporale:
      i risultati devono essere prodotti entro certi limiti temporali fissati (deadlines)
      (specifica dei sistemi real-time)

Sistemi Real Time: definizioni

- Elaborazioni Hard Real Time:
  il non rispetto delle deadlines temporali (missed deadline) NON e' ammesso
  - porterebbe a danneggiamento del sistema o di altri apparati o cose
    (safety critical) (complete or catastrophic system failure)
  - controllo digitale
- Elaborazioni Soft Real Time:
  il non rispetto delle deadlines (missed deadline) e' ammissibile
  - le specifiche temporali indicano solo in modo sommario i tempi da rispettare
  - degrado delle performace accettabile

Sistemi Real Time: definizioni

- Elaborazioni Hard Real Time:
  Esempi:
  - rilevamento di un overcurrent in un azionamento elettrico (possibile distruzione del convertitore di potenza)
  - allarmi in genere per sistemi complessi
  - Controllo in retroazione di tipo digitale su motori elettrici
- Elaborazioni Soft Real Time:
  Esempi:
  - risposta di un ascensore ad una chiamata
  - transizione dalla fase di inizializzazione alla fase di lavoro di una macchina automatica

Sistemi Real Time: definizioni

- In generale i sistemi real time complessi saranno ibridi hard/soft
  - Vi saranno elaborazioni con deadline inderogabili
    - es: gestione delle condizioni di allarme pericolose
    - controllo digitale in retroazione
  - Mentre altre elaborazioni avranno d.l. non stringenti
    - effettivo avviamento di una macchina dopo il comando ricevuto da un sistema di supervisione

Sistemi Real Time: definizioni

- Attenzione: spesso si confonde il concetto di Hard e Soft Real Time con quello di Real Time "Stretto" e "Largo"
- Real Time "Stretto":
  I vincoli temporali (deadlines) sono *stretti* rispetto ai tempi di calcolo necessari per eseguire le operazioni richieste
- Real Time "Largo":
  I vincoli temporali (deadlines) sono *larghi* rispetto ai tempi di calcolo necessari per eseguire le operazioni richieste

Sistemi Real Time: definizioni

- La "larghezza" o "strettezza" di un sistema di elaborazione dell'informazione real time dipende dalla piattaforma HardWare utilizzata
  - tempi di esecuzione dipendono dalla 'potenza' dell'unità di elaborazione (velocità + risorse)
- Purtroppo spesso i sistemi Hard Real Time sono anche stretti
  - normalmente le deadlines vicine sono anche inderogabili
  - per motivi di costo si sceglie sempre una piattaforma di elaborazione non sovrabbondante rispetto alle esigenze

Sistemi Real Time: definizioni

- Nel caso di sistemi REAL TIME STRETTI e' fondamentale la corretta organizzazione delle fasi di elaborazione per il rispetto delle deadlines
  - bisogna organizzare le attività di elaborazione al fine di ottimizzare i tempi
  - massimo sfruttamento delle risorse HW
    - forte legame tra il codice e l'HW di elaborazione
    - *difficile abbinare astrazione e ottimizzazione*.. ma non impossibile..

Applicazioni ed Eventi per i sistemi Real time

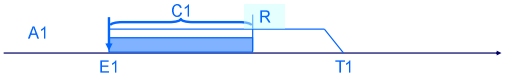
Sistemi Real Time: applicazioni ed eventi

- I sistemi real time devono in genere svolgere più applicazioni real time (*attività, task o processi RT*) che possono anche essere completamente indipendenti.
- Una attività RT (una volta avviata) al verificarsi di particolari eventi deve iniziare a compiere una elaborazione per fornire risposte prima che sia trascorso un certo tempo dall'evento
  - Deadline: istante entro cui dare la risposta
  - L'intervallo di tempo tra evento e deadline viene detto "time scope" (finestra temporale).

Classi di Attività

Attività di tipo Trasformazionale

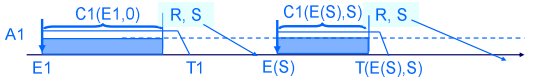
- L'attività viene avviata quando accade un certo evento
- Tutti i dati vengono rilevati dall'esterno all'avvio dell'applicazione
- Tutti i risultati/risposte vengono prodotti al termine dell'applicazione
  - La DEADLINE di risposta all'evento DIVENTA UNA DEADLINE SUL TEMPO DI DURATA DELL'APPLICAZIONE
  - ASSENZA DI INTERAZIONE CON L'ESTERNO DURANTE L'ESECUZIONE DELL'APPLICAZIONE (no altri eventi da gestire)
- Tipicamente l'insieme di operazioni eseguito è sempre il medesimo qualunque siano i dati di ingresso (carico comput. costante)



Classi di Attività

Attività di tipo Reattivo

- L'applicazione riceve eventi (e dati) e produce risposte durante tutto il tempo tra AVVIAMENTO e TERMINAZIONE
  - INTERAZIONE CONTINUA TRA AMBIENTE E APPLICAZIONE
  - PRESENZA DI DEAD LINE SULLE RISPOSTE NON NECESSARIAMENTE SULLA DURATA DELL'ATTIVITA'
- Tipicamente:
  - Data una risposta l'attività si pone in attesa di altro evento senza dover eseguire altre operazioni
  - L'evento che scatenerà l'elaborazione DIPENDE dalla storia passata (stato, S)
  - L'algoritmo eseguito per dare la risposta dipende dalla storia passata e dall'evento (carico comput. non costante)



Tipologie di eventi

- **Eventi Esterni o Interni al sistema di Elaborazione Real Time**
  - Es. evento esterno: interrupt provenienti dal 'campo'
  - Es. evento interno: scadenza di un timer
- **Eventi Periodici o Non Periodici**
  - Periodici:
    si ripetono ad intervalli fissi.
    - Campionamento delle variabili in un controllo digitale
  - Non Periodici, vengono classificati in:
    Aperiodici: si possono ripetere ad intervalli di tempo qualunque
    Sporadici: si possono ripetere ad intervalli di tempo superiori ad un certo lower bound.

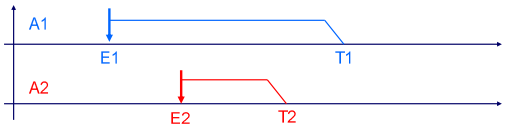
Ulteriore nomenclatura

- Task RT trasformazionali avviati da eventi periodici
  ⇒ task periodici o ciclici o ripetitivi
- Task RT trasformazionali avviati da eventi non periodici
  ⇒ task non periodici
- Per i task reattivi non si ha un qualcosa di analogo
  ⇒ gestiscono molteplici eventi.

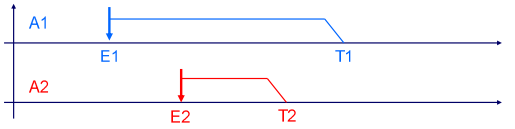
Problematiche di esecuzione parallela nei sistemi real time

Sistemi Real Time e Parallelismo

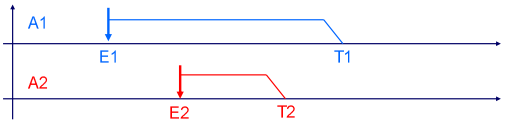
- I Sistemi Real time possono dover eseguire più attività RT
- I time scope relativi a eventi di diverse attività possono essere sovrapposti:
  ⇒ ESECUZIONE PARALLELA



Sistemi Real Time e Parallelismo



- La realizzazione parallela delle attività dipende dalla architettura HW utilizzata.
  - Sistemi monoprocessore (parallelismo logico, ma non reale)
  - Sistemi multiprocessore (parallelismo anche reale)
    - ravvicinati (collegamento molto veloce, es.: sulla stessa board)
    - distribuiti (collegamento più lento, es.: via rete ethernet)
- Parallelismo logico gestito con "Programmazione Concorrente"



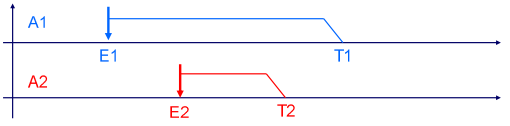
IN GENERALE LA PROGR. CONC. E' NECESSARIA SE:

- Numero di task paralleli > numero delle CPU
  - necessario gestire un PARALLELISMO LOGICO delle attività

E / O

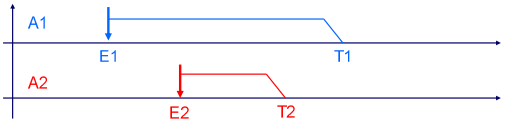
- Le diverse attività parallele agiscono su risorse condivise
  - es: memorie, periferiche di I/O

Tipicamente nei sistemi real-time entrambe le condizioni si verificano



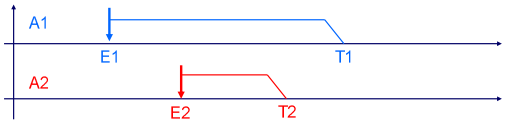
OBIETTIVO GENERALE DELLA PROGRAMMAZIONE CONCORRENTE:

- Fornire infrastruttura per
  GESTIRE PARALLELISMO LOGICO
  SENZA GESTIONE ESPLICITA NEI TASK
  - No "SW unico"
  - Portabilità
  - Scalabilità



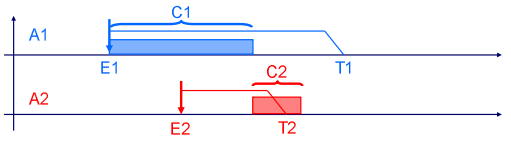
Elemento centrale della programmazione concorrente:

- ALGORITMI DI SCHEDULING
  Decide quale task eseguire effettivamente quando ce ne sono vari con time scope sovrapposti e/o risorse condivise
  Obiettivo particolare del RT: rispetto di tutte le deadlines
  Idealmente: ogni volta che si ha un nuovo evento bisogna che venga eseguito l'algoritmo di scheduling



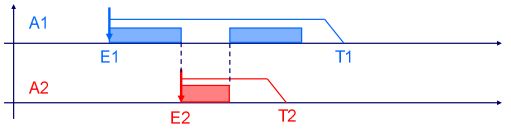
- Gli algoritmi di schedulazione influenzano pesantemente il rispetto delle deadlines
  - Esistono diversi algoritmi
- Vediamo Esempio

Esempio



- C_i tempo necessario all'esecuzione del task
  - dipende dal sistema di elaborazione
- Esempio:
  Sistema monoprocessore e Schedulazione "in serie"
  => NON SI RISPETTANO TUTTI I VINCOLI TEMPORALI

Esempio



- C_i tempo necessario all'esecuzione del task
  - dipende dal sistema di elaborazione
- Esempio:
  Sistema monoprocessore e
  Schedulazione "non in serie" (con Preemption) =>
  SI RISPETTANO TUTTI I VINCOLI TEMPORALI

Algoritmi di Scheduling per Sistemi RT

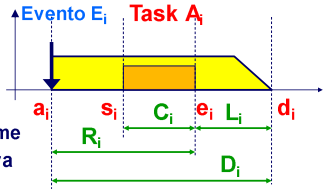
Definizioni

- **Scheduling (schedulazione):** ordinamento nel tempo delle elaborazioni richieste dai task su un certo parco CPU
- **Algoritmo di Scheduling:** algoritmo che produce una certa schedulazione dato un insieme di task e un insieme di CPU
- **Schedulabilità per sistemi RT:** insieme di task RT si dice schedulabile su una certa piattaforma HW (1 o più CPU) se esiste almeno un ordinamento nel tempo delle elaborazioni che *consente il rispetto di tutte le deadlines*

Algoritmi di Scheduling per Sistemi RT

Definizioni

- a_i: activation time
- d_i: deadline assoluta
- s_i: start time
- e_i: end time
- $C_i=e_i-s_i$ computation time
- $D_i=d_i-a_i$ deadline relativa
- $R_i=e_i-a_i$ response time
- $L_i=e_i-d_i$ lateness



Algoritmi di Scheduling per Sistemi RT

Definizioni:
CARATTERISTICHE DI UN GRUPPO DI TASK RT

- Task non interagenti
  - Gli *eventi* gestiti da un task sono *indipendenti* dagli altri tasks
  - No *vincoli di precedenza* o *comunicazioni bloccanti*
  - No *risorse condivise*
    - CPU a parte
- Task interagenti
  - Condizioni suddette violate
    - Anche una sola

Algoritmi di Scheduling per Sistemi RT

Definizioni:

STATI DI UN TASK RT RISPETTO AD ALGORITMI DI SCHEDULING

Un task RT (quando avviato e prima di terminare) può avere i seguenti stati:

- Inactive: non vi sono eventi pendenti da servire
  - Assenza eventi
  - Eventi già gestiti
- Active: vi sono eventi pendenti da gestire
  - Running: in esecuzione su CPU
  - Ready: in attesa di CPU
  - Blocked: bloccato in attesa di risorsa condivisa (no CPU) o vincolo di sincronizzazione
    - Significativo solo per tasks interagenti

Algoritmi di Scheduling per Sistemi RT

Problema generale

⇒ Definire algoritmi che trovino schedulazione conforme ai vincoli RT

- nel numero di casi più ampio possibile

- verificabile con test a priori

- *guaranteed scheduling*

- Punti critici:
  - TEMPO DI CALCOLO A RUN-TIME DEDICATO ALLA ESECUZIONE DEGLI ALGORITMI DI SCHEDULING
    - Riduzione del tempo CPU per task
  - QUANTITA' DI INFORMAZIONI CHE DEVONO ESSERE NOTE A PRIORI

Algoritmi di Scheduling per Sistemi RT

TIPOLOGIE DI ALGORITMI DI SCHEDULING

- Monoprocessore, Multiprocessore
- Preemptive, Non-preemptive
  - Task interrompibili
- Off-line: ordinamento dell'esecuzione definito a priori
  - Non solo ordine, ma anche istanti di cambio
- On-line: ordinamento definito a tempo di esecuzione in base a parametri attribuiti ai task
  - Priorità Statica: parametri (priorità) stabiliti a priori
  - Priorità Dinamica: parametri (priorità) variano durante l'esecuzione

Algoritmi di Scheduling per Sistemi RT

TIPOLOGIE DI ALGORITMI DI SCHEDULING

- **Off-line:** ordinamento dell'esecuzione definito a priori
  - Necessaria completa conoscenza a priori:
    - della "disposizione temporale" degli eventi
    - dei task
      - tempi di esecuzione esatti in ogni condizione e per ogni operazione
      - quando avvengono interazioni reciproche
  - Basso overhead a run-time
    - Tabella di azioni di schedulazione nel tempo definita offline
    - Sincronizzazione tra task interagenti risolta implicitamente
  - Scarsa Flessibilità, ma Elevata Predicibilità

Algoritmi di Scheduling per Sistemi RT

TIPOLOGIE DI ALGORITMI DI SCHEDULING

- **On-line:** ordinamento definito a tempo di esecuzione in base a parametri attribuiti ai task
  - Priorità Statica: parametri (priorità) stabiliti a priori
  - Priorità Dinamica: parametri (priorità) variano durante l'esecuzione
  - Minore conoscenza a priori (anche per test di schedulabilità)
    - No esatta conoscenza tempi di calcolo per ogni condizione
      - solo upper bound
    - No esatta conoscenza di quando e come avvengono interazioni
  - Overhead di calcolo a Run-Time
  - Maggiore flessibilità rispetto a Off-line

Algoritmi di Scheduling per Sistemi RT

CONSIDERAZIONI

- **Esistono diverse soluzioni**
  - No obiettivo di tale corso
  - Off-line: potenzialmente più performanti,
    - complessi (in caso generale NP-hard nel numero di task)
    - conoscenza a priori non sempre disponibile
  - On-line: algoritmi con diversi gradi di *efficacia*
    - *Efficacia*: capacità di trovare schedulazione conforme a vincoli RT quando esiste soluzione
    - In generale: algoritmi piu efficaci sono più pesanti!
      - Nell'implementazione possibile "DL missing" a causa di overhead di calcolo

Algoritmi di Scheduling per Sistemi RT

CONSIDERAZIONI

- Test di schedulabilità a priori:
  - Per off-line è l'algoritmo stesso
  - Per on-line molto complesso in generale
- Sono disponibili soluzioni off-line e on-line "trattabili" sotto ipotesi semplificative
  - SISTEMI MONOPROCESSORE
    - o multiprocessore con insieme task partizionato a priori
  - TASK TRASFORMAZIONALI CICLICI
    - con Periodi Ripetizioni T_i noti, costanti e Time-Scope$_i$=T_i
  - TASK NON INTERAGENTI
  - Scenario semplificato, ma interessante per automazione
  - Alcune soluzioni con riferimento a tali ipotesi

Algoritmi di Scheduling per Sistemi RT

ALCUNE SOLUZIONI

ON-LINE

- First In First Out (FIFO)
  - Solo per fini didattici
- Rate Monotonic Priority Ordering (RMPO)
- Earliest Deadline First (EDF)

OFF-LINE

- Timeline Scheduling (o Cyclic Executive)

(con riferimento alle ipotesi semplificative dette)

Algoritmi di Scheduling per Sistemi RT

ALCUNE SOLUZIONI: Preliminari

- Insieme di task come da ipotesi semplificative
- T_i= periodo di attivazione
- C_i= tempo di elaborazione
  - Max (se non è costante)
- U = utilization factor

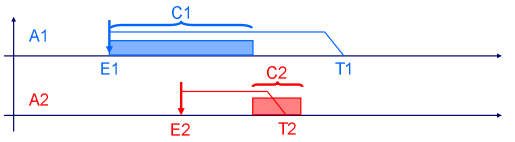
$$U = \sum_{i=1}^{n} \frac{C_i}{T_i}$$

- I termini C_i/T_i frazione di tempo CPU richiesta dal task i
- U rappresenta frazione di utilizzo di insieme
- Se U>1 insieme di task sicuramente non schedulabili con qualunque algoritmo

First In First Out

Ordinamento esecuzione secondo ordine di attivazione

- On line
- Priorità dinamica
- Non-preemptive
- Semplice
- Nessuna garanzia di rispetto DL



Rate Monotonic Priority Ordering

- On line
- Priorità statica
- Preemptive
- Ad ogni Task i-esimo viene assegnata priorità proporzionale a $1/T_i$
- Ogni qualvolta si attiva o disattiva un task
  - Attivazione = arriva evento ciclico che deve essere servito da un task
  - Disattivazione = task termina la sua elaborazione

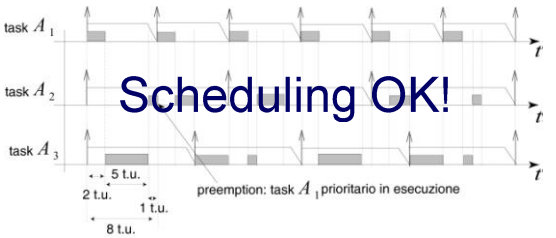
  ⇒ algoritmo RMPO analizza insieme dei task *attivi* e manda in *running* quello a priorità maggiore

RMPO: Esempio 1

Task	T	C	1/T	priorità
A_1	8	2	0.125	MAX
A_2	16	3	0.0625	MIN
A_3	12	5	0.083	MED

Fattore di utilizzazione

$$U_1 = \frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3}{T_3} \approx 0.8542$$

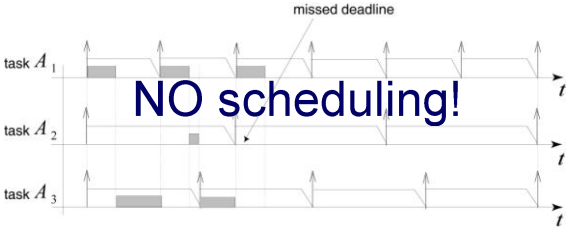


RMPO: Esempio 2

Task	T	C	1/T	priorità
A_1	8	3	0.125	MAX
A_2	16	3	0.0625	MIN
A_3	12	5	0.083	MED

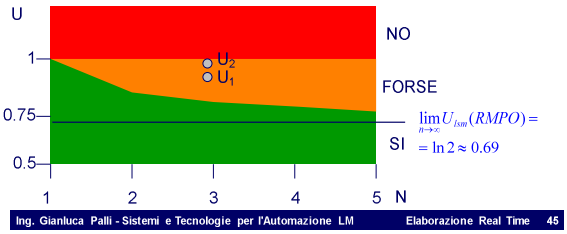
Fattore di utilizzazione

$$U_2 = \frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3}{T_3} \approx 0.98$$



RMPO: Test di schedulabilità

- Si può dimostrare che (nelle ipotesi dette):
  - RMPO è ottimo tra alg. on-line a priorità statica
  - Condizione sufficiente per schedulabilità per gruppo di n task: $U \le U_{lsm}(RMPO) = n(2^{1/n} - 1)$



Earliest Dead-Line First

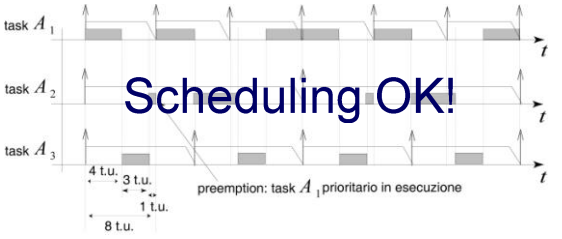
- On line
- Priorità dinamica
- Preemptive
- Ad ogni Task i-esimo *attivo* viene assegnata priorità inversamente proporzionale al tempo rimanente prima della D.L.
- Ogni qualvolta si attiva o disattiva un task
  ⇒ algoritmo EDF:
  - ricalcola la priorità dei task *attivi*
  - manda in *running* quello a priorità maggiore
    → i.e. quello con D.L. più vicina

EDF: Esempio 1

Task	T	C
A_1	8	4
A_2	24	6
A_3	12	3

Fattore di utilizzazione

$$U_1 = \frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3}{T_3} = 1$$

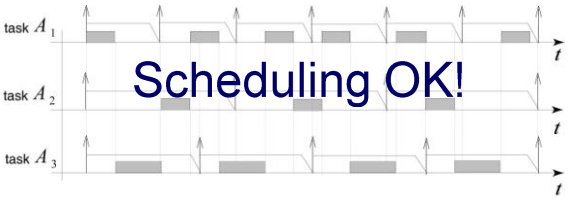


EDF: Esempio 2 (da Esempio 2, RMPO)

Task	T	C
A_1	8	3
A_2	16	3
A_3	12	5

Fattore di utilizzazione

$$U_2 = \frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3}{T_3} \approx 0.98$$



EDF: Test di schedulabilità

- Si può dimostrare che (nelle ipotesi dette):
  - EDF è ottimo tra alg. on-line a priorità dinamica
  - Condizione nec. e suff. per schedulabilità per gruppo di n task:

$$U \leq 1$$

  - Quindi EDF è ottimo rispetto a tutti gli algoritmi

RMPO vs. EDF

- EDF molto più efficace di RMPO
- RMPO computazionalmente più efficiente di EDF
  - Puo' risultare praticamente meno efficace
- Importante:
  Nell'ambito delle ipotesi semplificative dette, considerando **task *armonici***
  - *i.e. task ordinabili in modo che per ogni i $T_i = k_i T_{(i-1)}$ $k_i \in N$*

  RMPO per task con T_i *armonici* porta a stessa schedulazione di EDF.
  ⇒ Task armonici:
  si può abbinare efficacia ed efficienza

Timeline Scheduling (Cyclic Executive)

- Off-Line
- Non-preemptive
- Divisione del tempo in *Time Slices* (uguali) assegnate in modo fisso ai task
- Algoritmo
  - Si definiscono:
    - Major Cycle: m.c.m. dei T_i
    - Minor Cycle: M.C.D. dei T_i
  - Si fissa Time Slice = Minor Cycle
  - Ipotesi: tutti $C_i \leq$ Time Slice
  - Si prova ad assegnare ogni Time Slice in Major Cycle a Task o a sequenza di task (solo se $\sum C_i \leq$ Time Slice) in modo da rispettare le deadlines

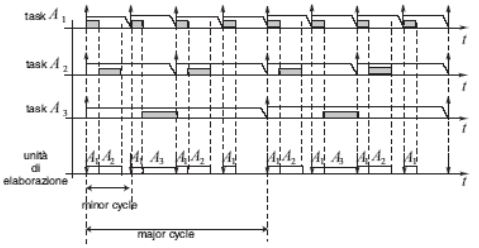
TS: Esempio

Task	T	C
A_1	4	1
A_2	8	2
A_3	16	3

Minor Cycle = 4
Major Cycle = 16

Fattore di utilizzazione

$$U_2 = \frac{1}{4} + \frac{2}{8} + \frac{3}{16} = \frac{11}{16}$$



TS: Considerazioni

- Svantaggi:
  - Non molto Efficace (vedi esempio)
    - Basso sfruttamento di CPU in generale (CPU più potenti)
- Vantaggi:
  - Determinismo assoluto
    - Tipico di off-line
  - Elevata Efficienza
    - Tipico di off-line e non-preemptive
  - Maggiore Affidabilità
    - Non-preemptive ⇒ no context switch ⇒ minimizzazione operazioni accessorie che possono dare errori
      - Minore possibilità di errori sw residui
      - Minore possibilità che bachi HW diano problemi non rilevabili

TS: Considerazioni

- Predicibilità ed Elevata Affidabilità
  - Proprietà molto importanti per sistemi safety-critical
    - i.e. failure può produrre danni seri
- Timeline Scheduling molto usato in:
  - Sistemi militari
  - Sistemi aeronautici
  - Controllo del traffico aereo
- Esistono anche varianti a maggiore flessibilità ed efficacia
  - Si usa preemption
  - Si perde parte dell'affidabilità (context switching)

Da On-Line a Off-Line?

- Con riferimento a scenario semplificato
- Per aumentare efficienza:
  - Valutare off-line i risultati di algoritmi on-line
    - Simulazione
  - Implementare l'ordinamento risultante secondo filosofia off-line
    - No algoritmi di scheduling a run-time
- Svantaggi:
  - Minore flessibiltà in presenza di:
    - Jitter su eventi di attivazione
    - C_i molto variabili
      - Taratura su caso pessimo
    - Task non RT o soft-RT eseguiti nei "tempi morti"

Relazioni tra i Sistemi di Controllo e i Sistemi Real Time

Due tipologie di attività svolte dai sistemi di controllo nel livello dei controlli della PA (vedi modello funzionale):

1) CONTROLLO DIRETTO DI VARIABILI TEMPORALI detto anche "controllo time-driven":
Controllo digitale in retroazione di sistemi dinamici tempocontinui (visto nei corsi precedenti)

- accezione "classica" del termine "controllo"
  - regolazione di temperatura di un forno
  - inseguimento di una traiettoria nello spazio di lavoro per un robot
  - controllo di assetto di un aeroplano
  - …

Due tipologie di attività svolte dai sistemi di controllo nel livello dei controlli della PA (vedi modello funzionale):

2) CONTROLLO DI SEQUENZE:
Supervisione e gestione delle diverse fasi di funzionamento dell'impianto da controllare

- tipicamente realizzata tramite una macchina a stati (automa) anche molto complessa
- in genere le attività di tipo 1 sono attivate dal sistema di supervisione in particolari fasi di funzionamento (i.e. in particolari stati)
- ambito metodologico: controllo di sistemi a stati finiti
  - area di ricerca…. sistemi ad eventi discreti, sistemi ibridi

Mappatura sulle attività Real Time

1) CONTROLLO DIRETTO DI VARIABILI TEMPORALI

⇓

Attività trasformazionale scatenata da evento ciclico:
campionamento delle variabili del plant

- Circa.. Attenzione: lo stato deve essere salvato!
  - Si potrebbe implementare con task che non termina, ma che diventa inactive per avere spazio dati di task permanente dove salvare stato (se non c'è mem. perman.)
  - Quindi mappatura non perfetta, ma è solo dettaglio "implementativo": è essenzialmente attività trasformazionale ciclica
    - Sempre stesso evento ciclico da servire
    - C costante
- Dette anche: attività data-flow

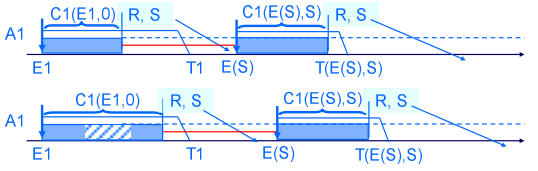
Mappatura sulle attività Real Time

2) CONTROLLO DI SEQUENZE:

⇓

attività "reattiva" interagente con eventi periodici e non

Peculiarità: legame tempo risposta e occorrenza evento successivo (sistema in feedback)

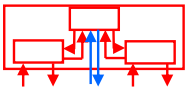


- **In generale controllori diretti di variabili temporali (R(z)) sono di tipo hard real time**
  Dead Line = Istante di campionamento successivo
  Time scope = tempo di campionamento
  - si può tentare di rilassare il vincolo hard compensando gli effetti dell'allungamento del Tempo di campionamento
    - riconfigurazione del controllore
    - comunicazione tra scheduler e algoritmo di controllo
    - difficile! Area di ricerca

- I controlli di sequenze sono tipicamente applicazioni di tipo soft real time
  - es: transito dalla fase di inizializzazione alla fase di run di una macchina automatica
  - Controesempio: gestione etichettatura bottiglia
- I sistemi per il controllo di impianti complessi presentano:
  parti di controllo diretto di variabili temporali,
  parti di controllo di sequenze
  Quindi vi sono vincoli hard e soft real time

Interazione tra attività RT di controllo

- Tipicamente ridotta al minimo (quasi assente)
  - Eventi serviti da Ctrl i-esimo non influenzati da ctrl k-esimo
    - Domini di controllo disgiunti
    - Quando ci può essere interazione spesso "risolta" con approccio implementativo
      - Tipicamente per ctrl logico "in cascata", ma ibrido



      - Vedi avanti per soluzione
        (O.S. RT time driven)

Interazione tra attività RT di controllo

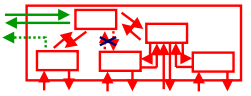
- Tipicamente ridotta al minimo (quasi assente)
  - Risorse condivise: NO
    - Sensori, Attuatori e Perferiche Comunicazione assegnati in modo esclusivo
      - Sensori e Attuatori appartengono ad un solo Ctrl
        - Assegnazione a priori immutabile
        - Assegnazione dinamica: sarebbe da evitare
          Nota: Motion Control (ind. manifatturiera) → si fa!
          se 2 Ctrl richiedono contemporaneamente a run time = errore grave di sistema (deve evitarlo progettista)
      - Periferica di comunicazione:
        - o Assegnata a unico Ctrl
        - o Task server con DL trattato come task di controllo
          Task controllo comunicano con task server come tra loro

Interazione tra attività RT di controllo

- Tipicamente ridotta al minimo (quasi assente)
  - Comunicazione e precedenze tra *controlli*:

    Comandi e misure "virtuali" in modello funzionale

    - Comunicazioni tipicamente NON BLOCCANTI
    - NO PRECEDENZE

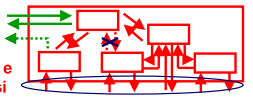
    

    - SE NO NUOVA COMUNICAZIONE
      → USA INFO VECCHIA E PROCEDI
    - Quando è importante comunicazione nuovo dato:
      → sincronizzazione a progetto di eventi (modello di esec.)
      → no gestione con bloccaggio su semafori e rischedulazione
      - Esempio: controllo in cascata con 2 tempi camp diversi
      - Spesso si usano peculiarità di O.S. (vedi avanti "timedriven")

Interazione tra attività RT di controllo

- Tipicamente ridotta al minimo (quasi assente)
  - Comunicazione tra *controlli e sensori/attuatori*:

    - Queste comunicazioni sono in genere MOLTO CRITICHE
    - Info da sensori e per attuatori devono spesso essere prodotte e consegnate in istanti ben precisi

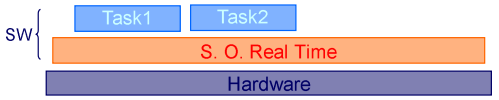
    ⇒ Ottenuto tipicamente con sincronizzazione da parte di progettista degli eventi che scatenano esecuzione task (come visto in slide prima: definizione opportuna del modello di esecuzione)

    No uso di strutture semaforiche con blocco assenza dati quando richiesti → generalmente errore grave (si sfruttano peculiarità di O.S. R.T. vedi avanti)

    NB: Può incidere su scelte di comunicazione via rete (sensori/attuatori su rete)

Sistemi Operativi per elaborazione Real Time Tipologia Time Driven e Event Driven

Architettura del HW-SW:



- S. O. Real Time:
  - gestione dell'I/O e delle periferiche
  - gestione delle immagini dei diversi tasks
  - rilevamento degli eventi e schedulazione dei tasks secondo algoritmi definiti per il real time
  - verifica del rispetto delle dead-lines a run-time
    - Misura e gestione del tempo
  - minimizzazione dell'overhead di S.O.

NB: sono in studio soluzioni innovative senza S.O., basate su "produzione automatica di codice", che massimizzano le prestazioni

Nota Bene:

- Il campionamento degli eventi e la gestione del tempo viene effettuata da S.O.
  - Task lavorano senza "coscienza diretta" del tempo
    - O meglio: POSSIBILITA' DI GESTIRE IL TEMPO
  - Progettati per dare i risultati nel minor tempo possibile
- In genere i task potrebbero essere interrotti da S.O. (sched preemptive) o potranno sospendersi autonomamente prima della terminazione
  - I Task Trasformazionali tipicamente non si sospendono autonomamente (arrivano a terminazione)
  - I Task Reattivi si sospenderanno autonomamente (inactive) ogniqualvolta si pongono in attesa di un evento
    - Verranno risvegliati dal sistema operativo
      - In modo diverso a seconda dell'approccio (time driven o event driven)

MODELLO DI ESECUZIONE

- Insieme di eventi/attività RT
  - Trasform. - Reattive
  - Tipicamente per ctrl:
    - Trasf. cicliche
    - PseudoTrasf. Cicliche
    - (vedi dopo)

+

- Unità di elaborazione
- O.S. Real time
  - Time/Event Driven (vedi dopo)
  - Algoritmo di schedulazione

⇓

MODELLO DI ESECUZIONE

Rappresentazione dell'effettivo svolgimento nel tempo dell'esecuzione delle attività real-time

MODELLO DI ESECUZIONE
NOTA BENE

- Modello di esecuzione:
  - Raffinamento della schedulazione (ordinamento temporale dato da set dei task e algoritmo di scheduling) che tiene conto anche delle caratteristiche e dell'overhead di O.S.
- Ovviamente il modello di esecuzione indotto dalle scelte implementative/tecnologiche deve (dovrebbe) soddisfare le caratteristiche funzionali (modello funzionale) richieste alle applicazioni RT
  - 1: Rispetto delle DL (correttezza temporale)
  - 2: Correttezza dei risultati

Due tipologie di approcci, a livello di SO, per rilevare gli eventi e schedulare i task:

- Approccio TIME-DRIVEN
- Approccio EVENT-DRIVEN

NB: tipicamente, in ambito informatico, la gestione dei task viene indicata come 'controllo dei task'

Approccio EVENT DRIVEN

- Ad ogni evento:
  - S.O. prende il controllo della CPU
  - S.O. analizza evento e decide quale/i task attivare
  - S.O. esegue algoritmo scheduling
  - S.O. manda in esecuzione il task scelto
- Approccio più intuitivo e che offre massime potenzialità, ma complesso realizzativamente
- Intrinsecamente legato a schedulazione on-line
  - Si considerano eventi non noti a priori

Approccio EVENT DRIVEN

- Considerazioni sulla complessità:
  - Essenzialmente: gestione a interrupt
  - Necessità di HW per la rilevazione di eventi e la generazione di interrupt
    - Eventi possono essere complessi
      - Combinazioni particolari di informazioni
    - Riconfigurabilità
      - Eventi da considerare legati alle applicazioni avviate
      - Possono variare a run time (appl. reattive)
  - Verifica delle deadlines
    - Contatori HW innescati e disattivati da OS
    - Producono interrupt se scadono..
    - Costo: numero di contatori "virtuali" pari almeno al numero massimo di time scope attivi

Approccio TIME-DRIVEN

- Non si reagisce al singolo evento, S.O. richiamato ad intervalli regolari di tempo per campionare la condizione di tutte le grandezze di interesse
- Si definisce tempo di scansione Ts:
  tempo che intercorre tra 2 campionamenti successivi
- Particolarmente adatto per schedulazione off-line
  - Eventi predefiniti
  - Allineamento degli intervalli Ts agli istanti significativi di schedulazione off-line
    - Esempio Timeline scheduling

Approccio TIME-DRIVEN

- Per gestione con algoritmi di schedulazione on-line ed eventi non predefiniti nel tempo
  ⇒ Attenzione:
  - Evitare la "perdita di eventi" ⇒ requisiti:
    1) Frequenza di scansione fs=1/Ts > fmax di evento
    2) L'HW per memorizzare l'accadimento di un evento fino al campionamento più prossimo
  - Ordine degli eventi nello stesso Ts non deve essere significativo
- Nel seguito ci si concentra su Time Driven associato ad algoritmi di sched. on-line ed eventi non preordinati

Approccio TIME-DRIVEN

- Essenzialmente: gestione a polling degli eventi
  - Opportuna scelta del Ts rispetto ai task
- Altre caratteristiche:

  Tipicamente: le risposte dei task vengono allineate agli istanti di campionamento del S.O.
  - Trasmissione/attuazione risposte gestite dal S.O.
    - Logico! Virtualizza gli I/O!
  - Tempi di risposta multipli di Ts

  Ma generalmente non solo….

Approccio TIME-DRIVEN

- Considerazioni:
  - Soluzione a polling rispetto a interrupt sposta riconoscimento degli eventi da HW a SW del SO
    - Più gestibile
    - Ma comunque complesso (eventi complessi e configurabili, anche a run time) ⇒ tempo di elaborazione
  - Campionamento ⇒ cambio di contesto anche inutile ⇒ tempo perso
  - Rimane la complessità degli algoritmi di scheduling
    - Tempo di elaborazione

  ⇒ elevato overhead di S.O. real time time driven

Approccio TIME-DRIVEN

- Generalmente in S.O. Real Time Time-Driven si introducono accorgimenti per:
  - Ridurre al minimo la complessità e tempo per riconoscere eventi
    - attività trasformazionali e reattive
  - Semplificare al massimo gli algoritmi di scheduling
    - mantenendo efficacia: alto grado di schedulabilità
    - vedi Time-driven multitasking

  Soprattutto nel mondo dell'automazione

Approccio TIME-DRIVEN: Accorgimenti per Applic. Trasformazionali Periodiche Note (es: "controllo diretto di variabili temporali")

- SINCRONIZZAZIONE tra campionamento S.O. e evento periodico

⇓

- Innesco del task senza bisogno verifica evento
- Minimizzazione overhead
- D.L.: il successivo inizio ciclo
  - Obbligatorio: vista la natura delle applicazioni trasf. periodiche

Approccio TIME-DRIVEN: Accorgimenti per Applic. Reattive (es: "controllo di sequenze")

- Ad ogni scansione il SO passa il campionamento delle grandezze al task
- Il task:
  - discrimina evento
  - aggiorna uscite e il proprio stato (se e' il caso)
  - Attesa nuovo evento auto sospensione
  - Tipicamente dead line inizio del Ts successivo
- Overhead: task attivato anche in assenza di evento
- Semplicità: riconoscimento evento non gestita da S.O.

Approccio TIME-DRIVEN: Accorgimenti per Applic. Reattive (es: "controllo di sequenze")

- Considerazioni sulla soluzione:
  - E' il task reattivo a polling sull'evento, non piu' il S.O.
    - Cambia implementazione del task
    - Ma non la sua definizione funzionale
  - L'applicazione reattiva è stata resa ciclica pseudo-trasformazionale sincronizzata alla scansione del S.O.
    - "pseudo"= carico variabile
  - Caso del Controllo Logico:
    corrisponde ad una implementazione sincrona dell'automa che rappresenta il controllo di sequenze
  - LA GESTIONE DEL TASK PER O.S. NON E' PIU' FORTEMENTE CASE-DEPENDENT

Approccio TIME-DRIVEN: Accorgimenti per Applic. Reattive (es: "controllo di sequenze")

- Altre considerazioni sulla soluzione:
  - Scelta di D.L. al successivo T_s
    - Non è obbligatoria!
    - Rende verifica (on line e a priori) del rispetto delle dead lines:
      - semplice
      - identica al caso delle applicazioni cicliche trasformazionali pure
    - Puo' portare a sovradimensionare la potenza di calcolo…
      - Vedi avanti

Approccio TIME-DRIVEN: Accorgimenti per Applic. Reattive (es: "controllo di sequenze")

- Altre considerazioni sulla soluzione:
  - Evento di attivazione task (non avviamento) è indipendente da plant
    ⇒ eliminazione di eventuale interazione temporale con altri task di controllo attraverso plant
      - Vedi considerazioni su interazione tra task tramite eventi

Diverse versioni TIME-DRIVEN:

- Time driven monotask
  - Più diffusa… PLC
- Time driven con multitasking
  - A singolo tempo di scansione… Diffusa: PLC
  - A più tempi di scansione

  ACCORGIMENTI PER SEMPLIFICARE LO SCHEDULING
    - Mantenedo l'efficacia

Approccio TIME-DRIVEN MONOTASK

- L'approccio time-driven è particolarmente adatto ad una programmazione MONOTASK
  - Su l'unità di elaborazione R.T. considerata si è deciso di implementare una sola funzione di controllo
    - Anche molto complessa…
- deadline unica e molto semplice:
  entro il campionamento successivo tutte le elaborazioni devono essere terminate

Approccio TIME-DRIVEN MONOTASK

- S.O. real time molto semplice:
  ciclo:
  - Inizio Ts: leggi input
  - Esegui task (entro la fine di Ts)
  - Fine Ts: Attua output

(*vedere esempio Arduino "EsempioEvoluto_TIME_DRIVEN"*)

- Tempo di risposta massimo rispetto eventi asincroni: 2Ts
  Ok per applicazione reattiva se time scope per la risposta ad ogni evento è ≥ 2Ts

Approccio TIME-DRIVEN MONOTASK

- Scelta di Ts:
  Caso di applicazione trasformazionale ciclica:
  Ts coincidente con il periodo di ripetizione dell'evento di innesco
  - Ovviamente occorrenza dell'evento e occorrenza della scansione di S.O. devono essere sincroni (già detto)
  - Controllo diretto di variabili temporali nel caso dell'automazione

Approccio TIME-DRIVEN MONOTASK

- Scelta di Ts:
  Caso di *applicazione reattiva mappata su pseudo-trasformazionale ciclica*:
  1) 2Ts ≤ Time Scope minimo
     - Verifica delle dead-line
     - Non critico: normalmente applicazione Soft RT
  2) Ts ≥ C_{max}
     - C_{max} massimo tempo di calcolo necessario
     - Riduzione interruzioni (non strettamente necessario)

Approccio TIME-DRIVEN MONOTASK

- Scelta di Ts: (continuazione)
  Caso di applicazione reattiva mappata su pseudo-trasformazionale ciclica:
  3) Ts ≤ Distanza minima tra 2 eventi da servire in sequenza
     - Di solito gratis per il controllo di sequenze:
       evento successivo può accadere solo dopo risposta al precedente

Approccio TIME-DRIVEN MULTITASKING

- Multitasking
  - Diverse funzioni di controllo sullo stesso sistema di elaborazione
- Diverse soluzioni:
  - a tempo di scansione unico
  - a più tempi di scansione (cicli comunque sincronizzati)
    - multipli interi uno dell'altro (es: 1-2-4, 1-3-6, 1-2-3)
      - Detti armonici
    - non multipli (es: 1-1.2-2.5, 1-2-3)

Approccio TIME-DRIVEN MULTITASKING

- A tempo di scansione unico
  - Il S. O. è analogo al caso monotasking:
    i task verranno serializzati e si dovrà garantire che nel tempo di ciclo tutti vengano eseguiti nel caso peggiore.
  - Semplice: scheduling statico seriale
  - Problema:
    Frequenza di scansione allineata al task più frequente
    Gestione non corretta per task che non richiedono tempi di intervento brevi e hanno carichi elevati
    - Controllo R(z) con Tcampionamento>>Ts
    - Gestioni di sequenze con latenze consentite >2Ts
    - Esempio…

Approccio TIME-DRIVEN MULTITASKING

- A più tempi di scansione
  - Per evitare "sovracampionamento" inutile si associano gruppi di task ai diversi tempi di scansione
  - più deadline da rispettare legate ai diversi tempi di scansione
  - schedulazione più complessa per verificare le deadline: tipicamente RMPO tra i gruppi associati ai Ts
    - tempi armonici:
      RMPO coincide con EDF: efficacia ed efficienza
    - tempi non armonici:
      schedulazione meno efficace
      GENERALMENTE EVITATA!

    Esempio… visto in precedenza

Approccio TIME-DRIVEN

- PRO
  - Gestione polling: HW più semplice
  - Possibilità di semplificare il riconoscimento degli eventi, sfruttando:
    - sincronizzazione con eventi periodici
    - mappatura delle attività reattive in cicliche pseudo trasf.
      - Attività rileva l'evento
  - Possibilità di usare schedulazione semplice a priorità statica
    - Singolo tempo di scansione
    - Tempi di scansione multipli armonici

Approccio TIME-DRIVEN

- PRO
  - Dai punti precedenti si ottiene:

  1) Semplicità del sistema operativo:
  Per ogni tempo di scansione:
  ciclo: lettura in - esecuzione task(s) - attuazione out
  unica deadline: prossimo istante di scansione

  2) Predicibilità: il massimo tempo di latenza rispetto ad un evento non sincrono è pari a 2 tempi di scansione.
    - Semplice verifica del rispetto delle deadline a priori
  - Altro possibile vantaggio: sincronizzazione per comunicazione a latenza "minima"
    - Stesso Ts: serializzazione per il data-path più critico
    - Diversi Ts: allineamento degli istanti per ridurre latenza
      - Esempio visto per controllo in cascata, diventa valido anche per ctrl logico

Approccio TIME-DRIVEN

- CONTRO
  - Mancanza di flessibilità:
    - la gestione di eventi sporadici non è ottimizzata (il task di gestione è sempre attivo)
    - uniformità di campionamento: tutti gli eventi sono campionati con la stessa frequenza
      - fs adattata all'evento più frequente
      - soluzione a tempi di ciclo multipli per ridurre il sovracampionamento inutile di alcuni eventi (armonici: per mantenere schedulazione semplice)
    - può portare a sovradimensionamento HW anche nel caso monotask
      - Varianti usate anche in PLC: si rilassa $Ts \geq C_{max}$
      - Prezzo: complicazione S.O.

EVENT-DRIVEN vs TIME-DRIVEN

- EVENT DRIVEN rispetto TIME DRIVEN:
  - può fornire prestazioni migliori
    - Maggiori gradi di libertà
  - complessità aumenta
    - Schedulabilità non garantibile a priori
      - Es: Applicazioni reattive o intereagenti
    - Comportamenti non completamente predicibili

PS: Regola generale
Più gradi di libertà
⇒ maggiori possibilità di migliorare
⇒ maggiori possibilità di sbagliare

EVENT-DRIVEN vs TIME-DRIVEN

- EVENT DRIVEN semplificato per controllo
  - Attività Reattive (controllo di sequenze) ridefinite come periodiche con certo T di ripetizione (simile a Time-Driven)
    - S.O. gestisce evento: inizio ciclo ripetizione, non eventi complessi
    - Differenza con time driven: No sincronizzazione con S.O. o altri task
  - Allora tutte le attività sono periodiche e indipendenti
    -
  - Con algoritmo di scheduling EDF è possibile verificare la schedulabilità a priori
  - Si ottiene buona flessibilità (non ottima) e garanzia
  - Attenzione: EDF pesante ⇒ overhead di O.S. L'efficienza complessiva potrebbe essere bassa!

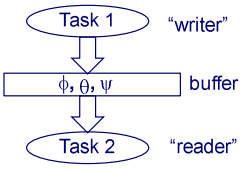
Nota Bene

- Obiettivo di questa parte: Sistemi RT per il controllo
  - Ci si è concentrati su
    applicazioni cicliche trasformazionali (controlli digitali)
    applicazioni reattive (controllo di sequenze)
    - Generalmente rimappate in cicliche pseudo-trasformazionali
  - Sono quelle tipiche per i controlli
  - Nei sistemi RT generici sono comuni anche le attività trasformazionali aperiodiche (qualche volta anche nel controllo)
    - non considerate
    - spesso trattate in modo simile alle reattive: rimappate su cicliche pseudotrasformazionali
    - per approfondimenti si rimanda ai libri consigliati

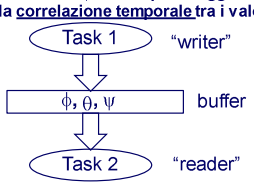
Servizi di Sistema per l'implementazione di Applicazioni Real-Time su RTOS

- Modello ideale visto prima:
  - task non interagenti, preemption eseguibile in qualunque momento
- Modello reale:
  - possibile comunicazione fra tasks
  - possibile accesso a risorse condivise
    - e.g. porte di comunicazione, etc.
- Funzioni del S.O. real-time (RTOS)
  - comunicazione
  - semafori per gestione risorse condivise
    - problema del deadlock
    - problema della inversione di priorità
  - servizi per la gestione del tempo

- Comunicazione
  - Spesso si utilizza uno spazio di memoria comune tra itasks, ma attenzione!
  - Esempio: controllo di assetto
    - Task 1: legge accelerometri e giroscopi e calcola l'assetto del veicolo (angoli ϕ, θ, ψ)
    - Task 2: legge l'assetto del veicolo e calcola l'azione di controllo (R(z)) per stabilizzare il veicolo

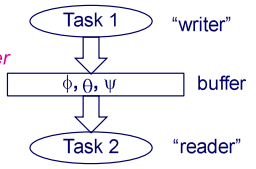


- Comunicazione
  - Esempio: controllo di assetto
    - Possibile esecuzione errata:
      - Task 1 scrive ϕ e θ ma poi avviene la preemption ed esegue Task 2
      - Il valore di ψ non è quindi aggiornato! non è rispettata la correlazione temporale tra i valori nel buffer

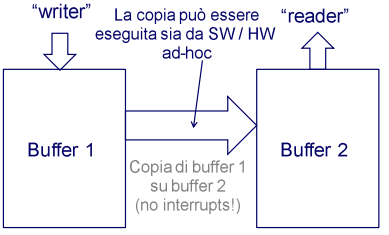


- Comunicazione
  - Esempio: controllo di assetto
  - Soluzione: rendere "atomica" la scrittura di ϕ, θ e ψ sul buffer

```
TASK 1
....
buffer.clean(); //clean buffer memory
disable_interrupts()
buffer.add(phi);  //add phi to buffer
buffer.add(theta); //add theta to buffer
buffer.add(psi); //add psi to buffer
enable_interrupts();
....
```



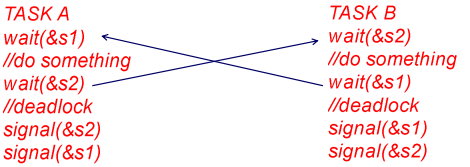
- Comunicazione
  - Il S.O. può mettere a disposizione meccanismi ad-hoc
  - Esempio:
    - DOUBLE BUFFERING (due indentici buffer)



- Gestione Risorse Condivise: Semafori
  - DEF: Sezione critica: parte di codice da eseguire in maniera mutualmente esclusiva
  - DEF: variabile semaforica s
    - variabile binaria 1: libero 0: occupato
    - due operazioni:
      - wait(&s):
        - se s=1 continua e setta il semaforo a 0 (GET LOCK)
        - se s=0 bloccante
      - signal(&s):
        - setta s al valore 1 di libero (RELEASE LOCK)

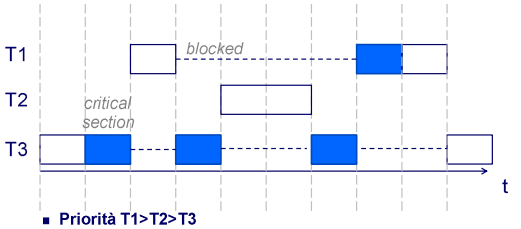
```
wait(&s)
//critical region
signal(&s)
```

- Gestione Risorse Condivise: Potenziali Problemi
  - Deadlock



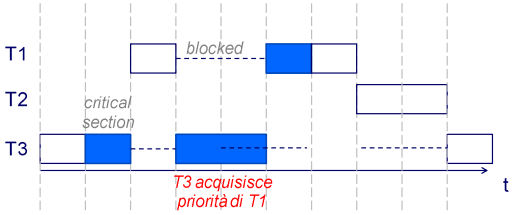
  - Ipotesi: A e B sono preemptive
    - allora A può eseguire la wait() su s1 e poi lasciare la CPU a B che esegue la wait() su s2
    - a questo punto né A né B possono più andare avanti (DEADLOCK)

- Gestione Risorse Condivise: Potenziali Problemi
  - Inversione di priorità



  - Priorità T1>T2>T3
  - però T3 blocca una risorsa condivisa con T1. Conseguenza: T2 che ha priorità > T3 ma < T1 esegue come fosse il task con priorità maggiore!

- Per evitare il fenomeno dell'inversione di priorità incontrollata le primitive che operano sui semafori realizzano il protocollo d'accesso alle risorse condivise Priority Inheritance



  - T3 blocca una risorsa condivisa con T1 ma ne eredita la priorità. Conseguenza: T3 conclude le proprie operazioni come fosse il task con priorità maggiore sbloccando T1

- Alcune regole generali per minimizzare problemi di sincronizzazione fra tasks
  - Minimizzare numero e lunghezza di sezioni critiche
  - Rilasciare il LOCK il più velocemente possibile
  - Evitare preemption dei tasks all'interno delle sezioni critiche
  - Le sezioni critiche sono parti di codice molto delicate, deveno essere opportunamente validate!

- Servizi per la gestione del tempo
  - Alcune caratteristiche desiderate
    - precisione
      - operazioni di integrazione nelle R(z), etc.
    - primitive (system calls) per gestione temporale dei tasks
    - contatori e rilevazione delle missed deadline
- Servizi per la gestione della memoria (cenni)
  - servizi per gestire il cambio di contesto
    - uno o più runtime stacks
    - gestione mediate task control block
      - gestione mediante due liste: 1) lista tasks sospesi 2) lista tasks ready

- Scelta del S.O. real-time
  - Alcuni parametri da considerare attentamente:
    - "minimum interrupt latency"
    - numero massimo consentito di tasks
    - "memory requirements" (dimensione immagine del S.O. in memoria) e servizi per la gestione della memoria
    - scheduling policies (FIFO, RMPO, EDF)
    - meccanismi per la comunicazione e sincronizzazione dei tasks
    - supporto alle applicazioni, supporto CPU (x86, ARM vXX, etc.), disponibilità codici sorgenti

Laurea Magistrale in Ingegneria Informatica
Laurea Magistrale in Ingegneria Elettronica e Telecomunicazioni per lo Sviluppo Sostenibile

Sistemi e Tecnologie per l'Automazione LM

Sistemi di elaborazione Real Time per l'Automazione

FINE

Ing. Gianluca Palli
DEI - Università di Bologna
Tel. 051-2093186
E-mail: gianluca.palli@unibo.it
http://www-lar.deis.unibo.it/people/gpalli/