No. SISTEMI INF.

Date 18 09 19

Presentazione Corso

Esercitatrice: Giulia Romano

Libro (consigliato): Sistemi in tempo reale, Giorgio Buttazzo

Materiale: Beep → esercitazioni, temi d'esame

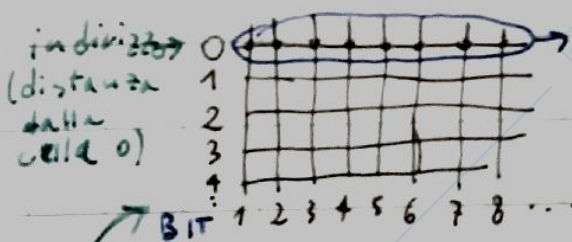
Esame: no prove in itinere

struttura esame:
1. Statechart (10 pt.)
2. Es. di scheduling real time (8 pt.)
3. Domanda teorica (6 pt.)
4. Codice Java (8 pt.)

INIZIO LEZIONI

ORGANIZZAZIONE CALCOLATORE

Memoria Centrale

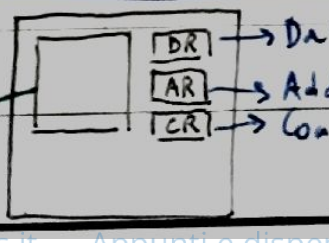


→ cella di memoria

tutte le celle hanno lo stesso numero di bit

→ Lettura e scrittura da memoria

il numero di bit in Data Register è uguale a questo



DR → Data Register
AR → Address Register
CR → Control Register

Registri

Sono connessi con il bus di sistema

Lettura

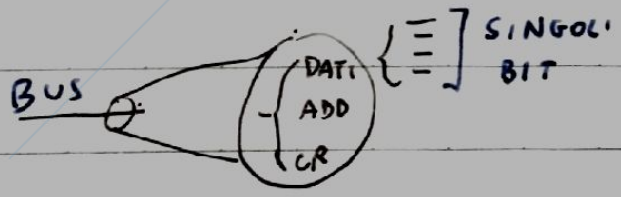
1. CPU → AR : individua cella da cui leggere
2. CPU → CR (leggi) : inserisce il comando leggi
3. DATO PUNTATO DA AR → DR : legge il dato e lo mette in DR

Scrittura

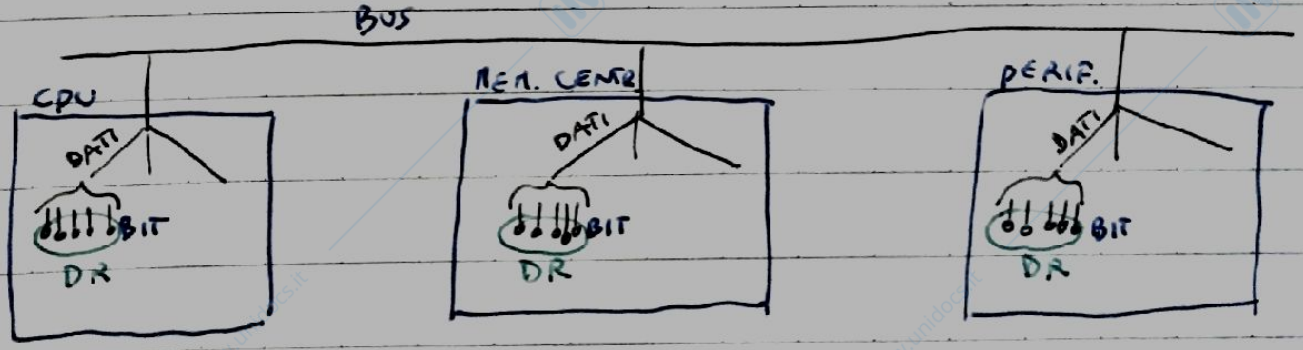
1. CPU → AR : individua cella su cui scrivere
2. CPU → DR : contemporaneamente prende il dato da scrivere
3. CPU → CR (scrivi) : inserisce il comando scrivi

Bus

3 linee di connessione



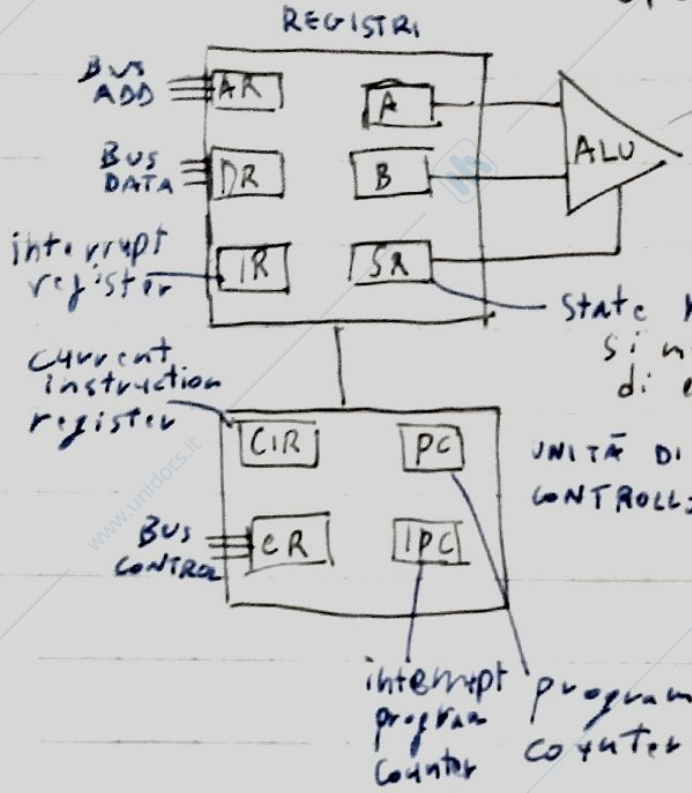
L'accesso al bus è totalmente regolata dalla CPU



Il bus è condiviso, quindi se si mette un'informazione sul bus tutti quanti la vedono.

Cpu

CPU



Unità algebrico-logica

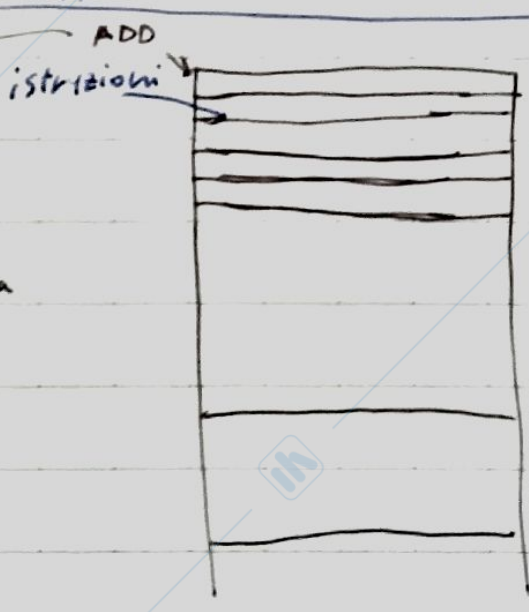
operazioni aritmetiche, binarie

A A e B

State register: si mette il segnale di errore dovuto alle operazioni

guarda l'istruzione e per ogni istruzione mette il codice che prenderà messo sul bus.

Come viene rappresentato un programma in memoria?



alla CPU il programma appare come l'indirizzo della cella da cui parte il codice

dentro al CIR è contenuto il contenuto della cella di memoria corrente

dentro al PC è contenuto l'indirizzo della cella con la prossima istruzione

→ per eseguire un programma mettiamo in PC l'indirizzo della prima cella

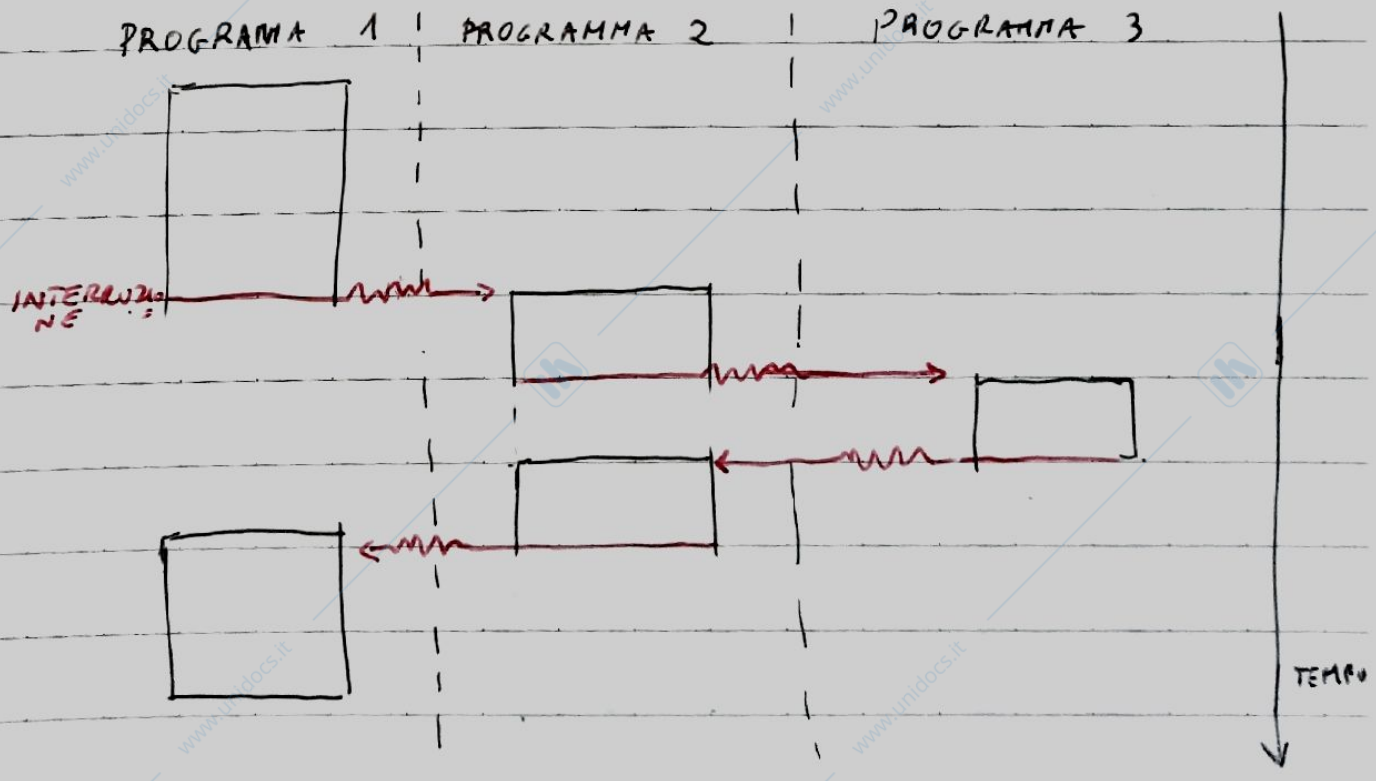
Ad ogni giro di clock la memoria controlla l'indirizzo su PC

→ Esecuzione programma:

1. ADD. PRIMA CELLA → PC
2. PC → AR
3. LETTURA MEMORIA
4. DR → CIR
5. ESECUZIONE CIR

Dopo che il contenuto è stato caricato in CIR, PC viene automaticamente incrementato e va all'indirizzo successivo.

Interruzioni



5

Mo Tu We Th Fr Sa Su

No. SISTEMI INF.

Date 18 · 09 · 19

1. Segnale di interruzione

↓

INTERRUPT → ad ogni ciclo di clock la CPU vede se c'è un INTERRUPT nell'IR

↓

IR

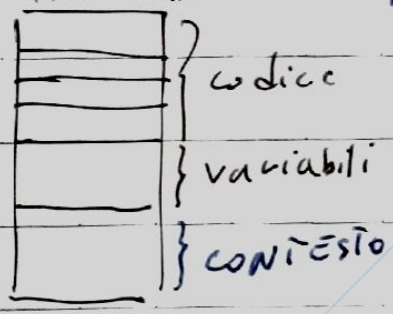
2. IR ⟶ ISR

interrupt subroutine

↓

programma

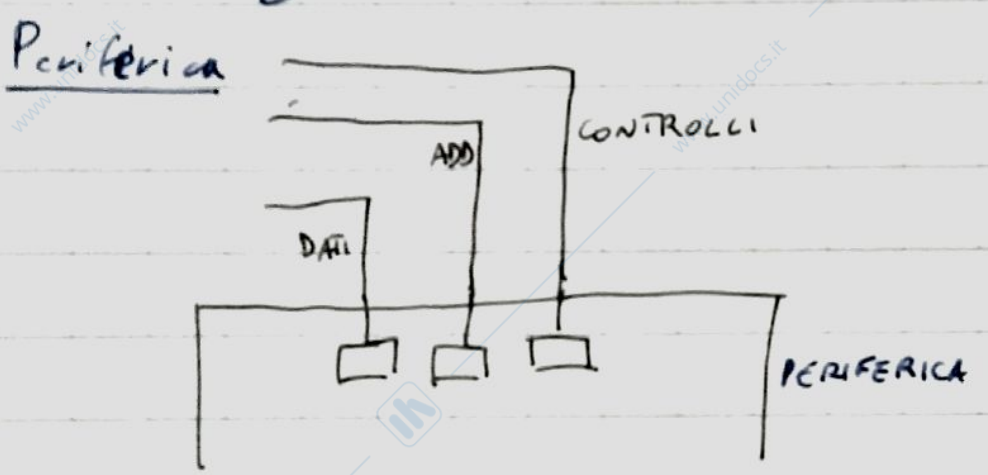
Prima di ritornare sul programma interrotto dal programma su cui siamo andati dopo l'interruzione bisogna salvare il CONTESTO: contenuto originale dei registri prima che il programma venisse interrotto



mascherabili poiché possiamo ignorare alcuni interrupt

Tipologia degli interrupt

- HARDWARE
  - MASCHERABILI (IRQ): della periferica
  - NON MASCHERABILI (NMI): generato dalla macchina
- SOFTWARE: generato da un programma in esecuzione



La gestione della periferica avviene esattamente come quella della memoria centrale

→ Gestione dell'input/output

1. POLLING: metodo con cui controlli periodicamente con un programma, interrogando la periferica, cosa è avvenuto

2. INTERRUPT: metodo per cui se accade qualcosa la periferica ci segnala con un interrupt che è accaduto l'evento

Sembra meglio del 1° metodo poiché non devo sempre interrogare la periferica

→ ma la CPU potrebbe essere completamente bloccata per gestire tanti interrupt se stanno accadendo tanti eventi

7

Mo Tu We Th Fr Sa Su

No. SISTEMI INF

Date 18 · 09 · 19

3. DIRECT MEMORY ACCESS (DMA): metodo per cui non è necessario che la CPU gestisca la periferica, che può direttamente accedere (trasferire informazioni controllando il bus) alla memoria centr.

↓

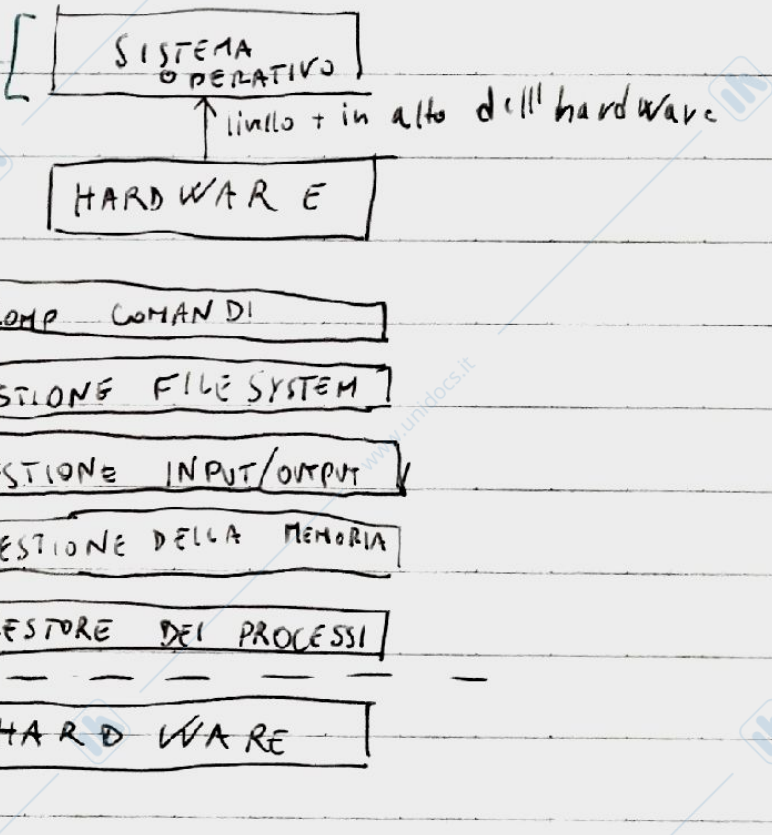
permette che la CPU svolga altro → la CPU non deve tenere la sua CACHE durante l'operazione

A senso usare il DMA quando abbiamo grandi flussi da tastiera dalla periferica alla memoria.

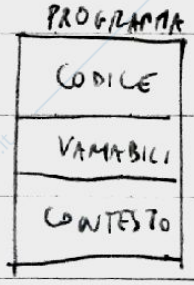
|

nei sistemi critici è rischioso usarlo.

Sistema operativo



Processo



PROCESSO = (CODICE, VARIABILI, CONTESTO)

! programma in esecuzione con tutti i dati che descrivo lo stato corrente del programma

2

Mo Tu We Th Fr Sa Su

No. SISTEMI INF

Date 19 · 09 · 19

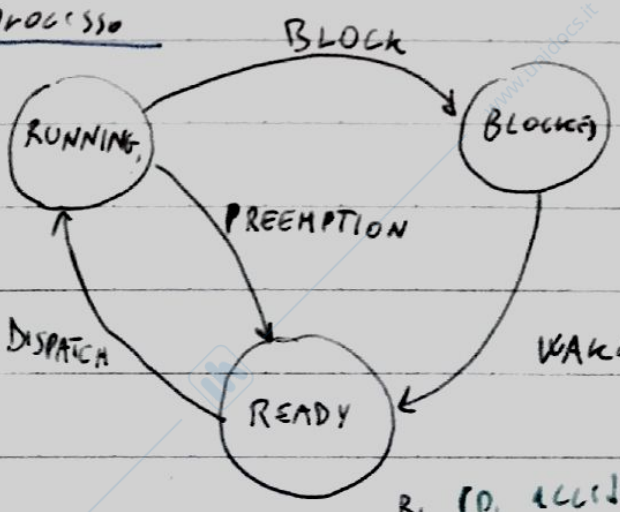
Consideriamo dei processi $P_1, P_2, \ldots, P_n$

Per ogni processo abbiamo uno stato che può essere di tre tipi

→ STATO: - IN ESECUZIONE (RUNNING SUL CIR)
- PRONTO (READY)
- BLOCCATO (BLOCKED)

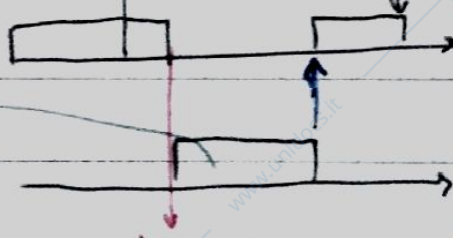
Un processo può avere bisogno di Risorse: - mutuamente esclusive: può essere utilizzata da un processo alla volta
es. periferiche
- non mutuamente esclusive

Cicli di un processo



R_1 (P_1 accede alla risorsa R_1)

P_2 cerca l'accesso a R_1 ma non può → Gestore dei processi lo blocca e manda P_1 in esecuzione

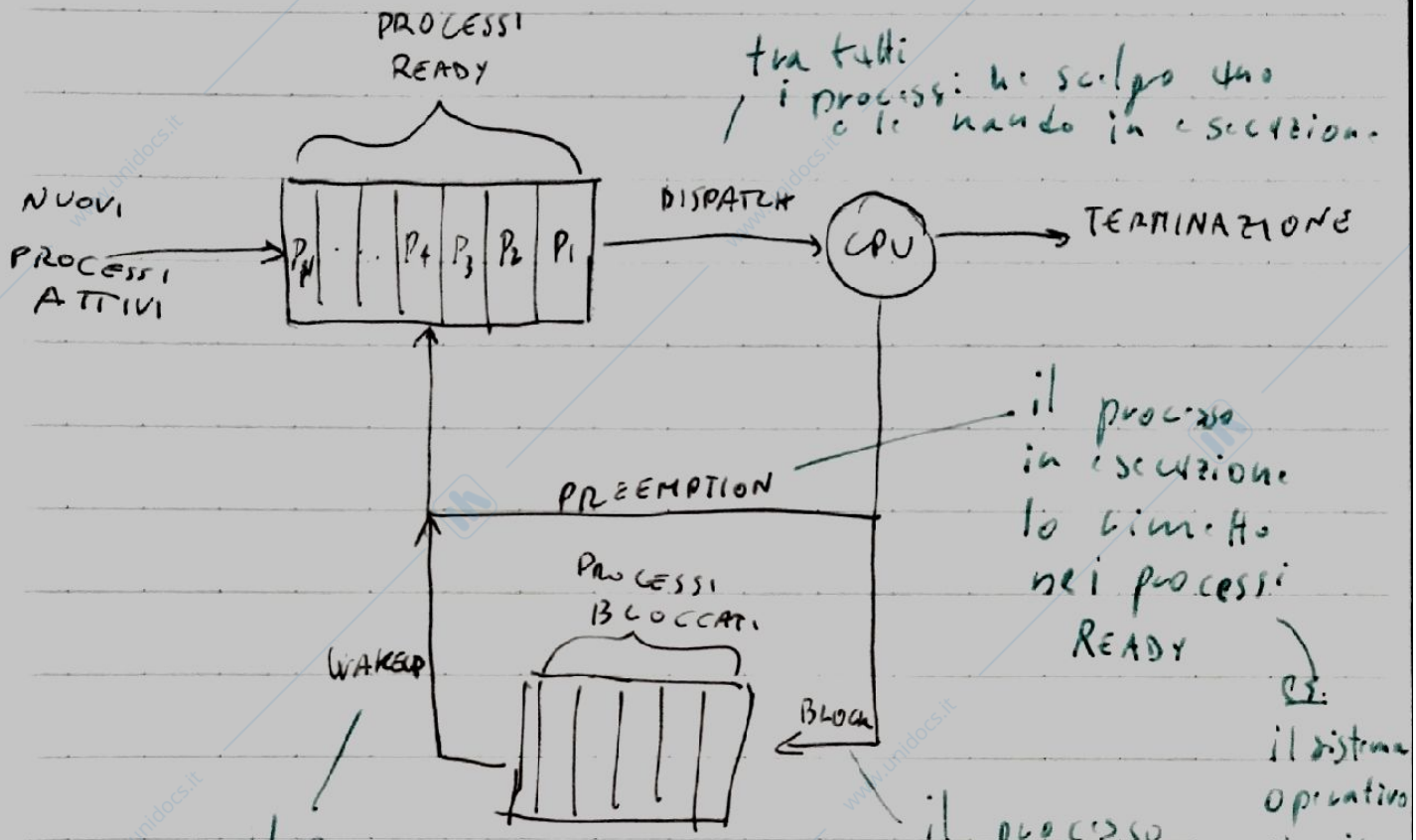


rilascia R_1 e P_2 va in READY

interrupt viene gestito da P_2

P_2 in esecuzione
P_1 in ready

La gestione dei processi avviene in modo che da BLOCKED NON passi direttamente a RUNNING e da READY NON passi direttamente a BLOCKED.



P.S. il sistema operativo decide di fermare un processo per eseguirne un altro

es. Il sistema decide di dare lo stesso tempo di esecuzione a $P_1, P_2 \dots P_N$ quindi interrompe P_1 per dare lo stesso tempo agli altri → QUANTI TEMPORALI

anche se non c'è nessuna risorsa bloccata

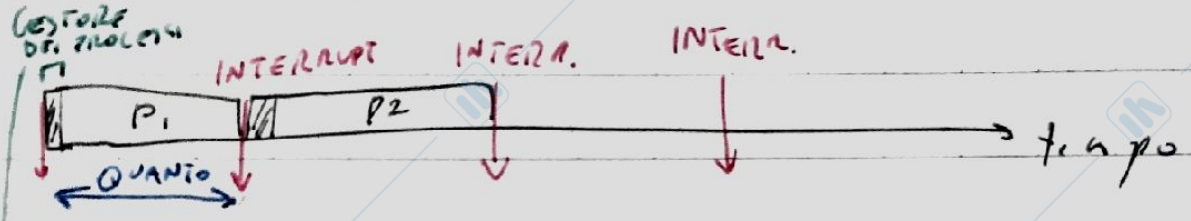
4

No. SISTEMI INF.

Mo Tu We Th Fr Sa Su

Date 19 · 09 · 19

Quanti temporali



il gestore dei processi può decidere allo scadere del quanto di tempo che processo attivare

→ Ad ogni interrupt riprende i processi READY e sceglie quale eseguire e poi lo manda in esecuzione
(il tempo che impiega per fare questo è trascurabile)

[Più i quanti sono grandi, più è difficile rispondere agli interrupt, ma non possono nemmeno essere troppo piccoli altrimenti il tempo impiegato dal gestore dei processi non diventa più trascurabile.]

5

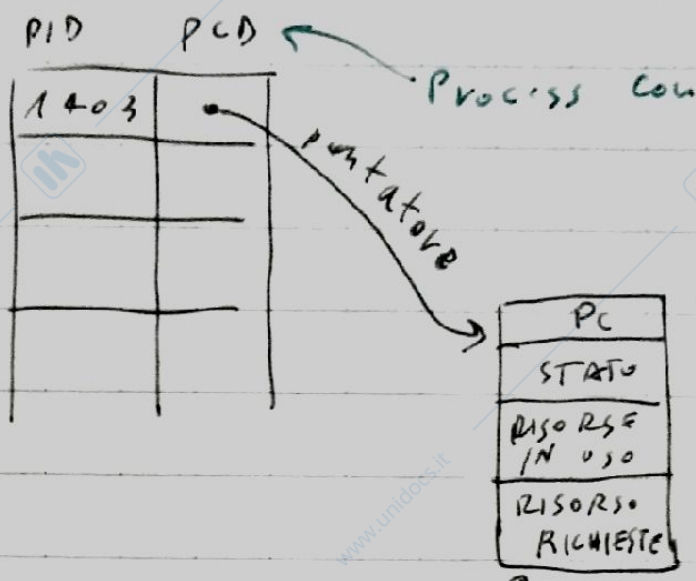
Mo Tu We Th Fr Sa Su

No. SISTEMI INF.

Date 19 · 09 · 19

MEMORIA CENTRALE

Process Identifier: numero che identifica il processo



Per ogni processo ha i queste informazioni che vengono esaminate dal gestore dei processi ogni volta che li deve gestire.

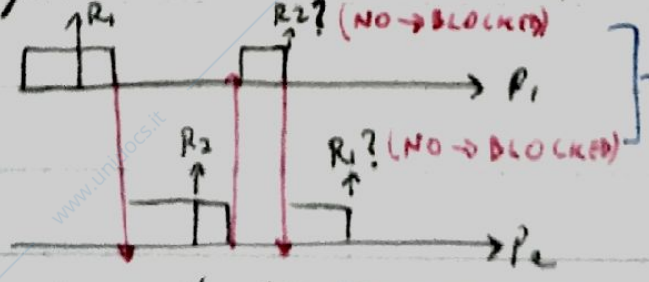
Scheduling

È il problema di decidere in quale successione vanno eseguiti i processi.

È un problema che cambia nel tempo, ma non sappiamo quando e come.

Concorrenza

Immaginiamo di avere 2 processi e 2 risorse



→ non possiamo eseguire né P1 né P2 poiché entrambe le risorse che gli servono non sono disponibili

Impedisce al sistema di proseguire ← DEADLOCK

6

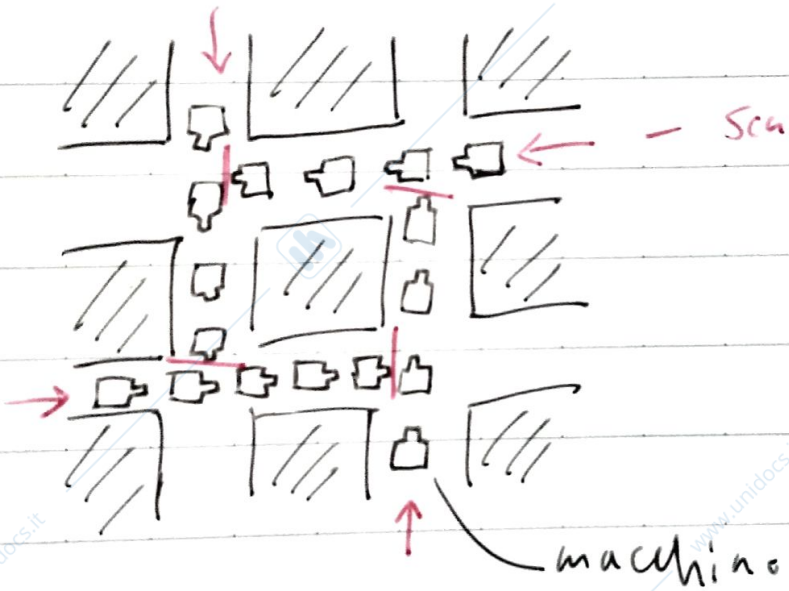
Mo Tu We Th Fr Sa Su

No. SISTEMI INF.

Date 19 · 09 · 19

DEADLOCK

Possiamo immaginarla come se in un incrocio:



Il modo per evitare le deadlock è verificare e certificare prima dell'esecuzione dei processi che non si verifichi una DEADLOCK.

REAL TIME

⟹ SCHEDULARE PROCESSI SODDISFACENDO VINCOLI (Scadenze temporali, ...)

- PROCESSI
  - PERIODICO
  - APERIODICO
- caratterizzati da
  - SCADENZE TEMPORALI
  - COSTO COMPUTAZIONALE
  - TEMPO DI ATTIVAZIONE → nel caso di periodici tempo di attivazione periodico
  - (RISORSE MUTUAMENTE ESCLUSIVE)
  - VINCOLI DI PRECEDENZA

→ con queste informazioni vogliamo trovare una schedulazione

$f: T \to P$

→ è un problema di OTTIMIZZAZIONE

2

No SISTEMI INF.

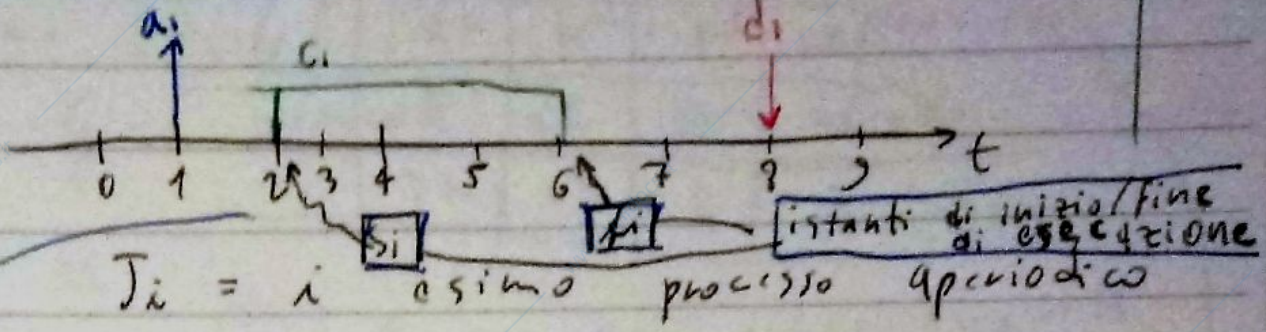
Date 20 09 19

Modello

Dobbiamo innanzitutto modellare i processi:

→ Processo Aperiodico

parametri in uscita



J_i = i esimo processo aperiodico

	J_1	J_2	J_3
a_i	1		
c_i	4		
d_i	8		

← INPUT = MATRICE

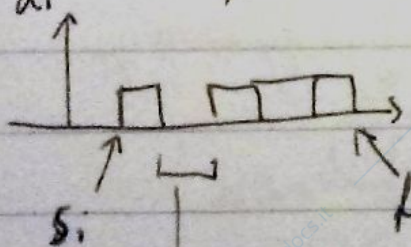
tempo di attivazione → quanto ci mette ad andare in READY

costo computazionale

scadenza temporale (Deadline assoluta)

Tempo di risposta:

$R_i = f_i - a_i$



anche se c'è un'interruzione s_i e f_i rimangono questi

Deadline

ASSOLUTA – RELATIVA

si indica con D_i

es – la lezione deve finire alle 13:00 (ASSOLUTA)

vs

– la lezione deve durare un'ora e mezza (RELATIVA) → si deve calcolare relativamente a quando

Immaginiamo che la matrice di prima sia completata

	J_1	J_2	J_3
a_i	1	1	2
c_i	4	2	3
d_i	8	10	15

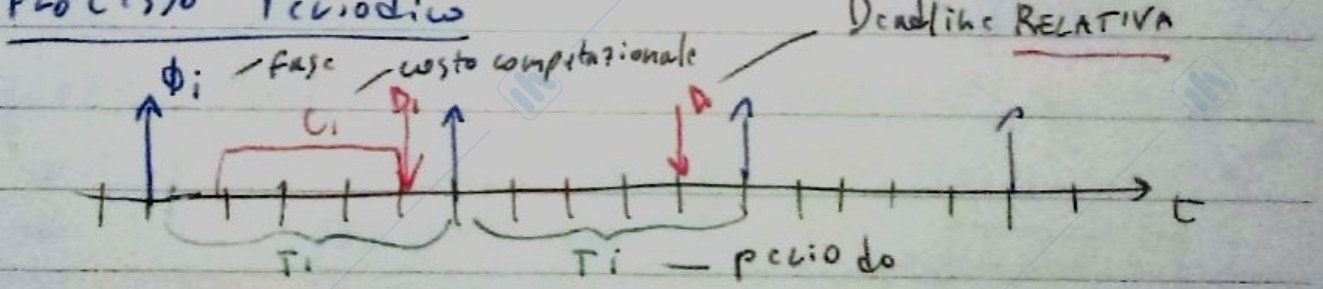
→ All'istante 0 il gestore dei processi non vede nessun processo

→ All'istante 1 vede che sono in READY il processo J_1 e J_2 (la matrice si rivela all'algoritmo piano piano nel tempo)

↑

L'INPUT è disponibile in modo ON-LINE

→ Processo Periodico



$\tau_{i=i}$ esimo processo periodico

	τ_1	τ_2	τ_3
ϕ_i			
T_i			
C_i			
D_i			

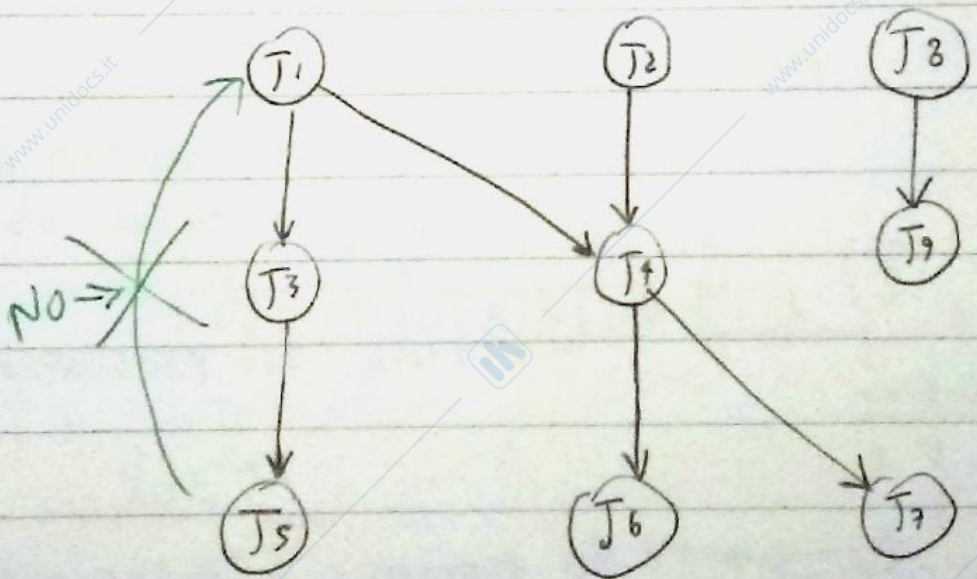
$\tau_{i,k}$ = k-esima attivazione del processo periodico i-esimo

Chiamiamo $(\mathcal{T}, \mathcal{C})$ l'insieme dei processi e $\mathcal{G}$ i vincoli di precedenza

es.

J_2 non può essere eseguito prima della terminazione di J_1

Se abbiamo tanti processi

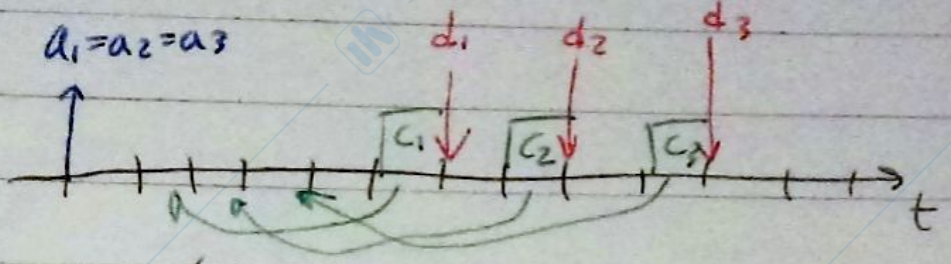


Questo grafo rappresenta i vincoli di precedenza, può essere sconnesso ed è un DGA = Direct Acyclic Graph

non ci possono essere cicli

INSIEME PROCESSI — VINCOLI DI PRECEDENZA

Abbiamo $(T, \mathcal{C})$, G può ci serve altro, prendiamo



$C_1 = C_2 = C_3 = 1$

potremmo metterli prima così finirebbero prima della deadline

↓

SOLUZIONE + ROBUSTA: vogliamo metterli il + possibile lontani dalla deadline

— funzione obiettivo

↓

massimizza la funzione obiettivo

↓

$(T, \mathcal{C}), G, M$ — MISURA DI PRESTAZIONE

Funzioni obiettivo

1. $R = \sum_i \frac{R_i}{⑤}$ — numero dei processi

R ↓ tempo di risposta medio

← FUNZIONE OBIETTIVO: MINIMIZZARLA

2.

$L_i = f_i - d_i$

lateness

OBIETTIVO: MINIMIZZARLA

Osserviamo che

$L_i > 0 \rightarrow f_i > d_i \rightarrow$ DEADLINE VIOLATA

$L_i = 0 \rightarrow f_i = d_i \rightarrow$ TERMINAZIONE ALLA DEADLINE

$L_i < 0 \rightarrow f_i < d_i \rightarrow$ HO MARGINE PRIMA DELLA DEADLINE

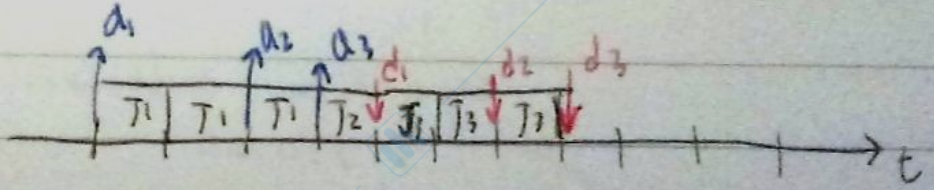
lateness massima

$\rightarrow L_{MAX} = MAX\{L_i\} \rightarrow$ VA MINIMIZZATA

FUNZIONE MASSIMO RITARDO

(es)

Consideriamo:



$L_1 = -1$

$L_2 = -2$

$L_3 = 0$

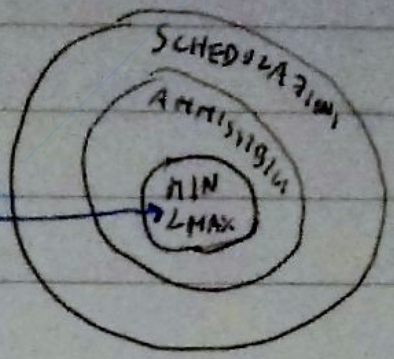
$\rightarrow L_{max} = 0$

Mo Tu We Th Fr Sa Su

No. SISTEMI INF.

Date 20 · 09 · 19

OTTIMIZZARE



Scelgo questa soluzione

↓

possiamo prendere semplicemente quelle con L_{max} minimo poiché anche non guardando i vincoli, sta già nelle ammissibili, che significa che rispetta i vincoli.

↘ ROBUSTA

$$N_{late} = \sum_i MISS_i(f_i)$$

N_{late}: NUMERO PROCESSI IN RITARDO

$$MISS(f_i) = \begin{cases} 1 \text{ se } f_i > d_i \\ 0 \text{ ALTRIMENTI} \end{cases}$$

OBIETTIVO: MINIMIZZARE

↓

significa soddisfare i vincoli

molto meno ROBUSTA

Se non ammette soluzioni ammissibili si possono scegliere vari criteri