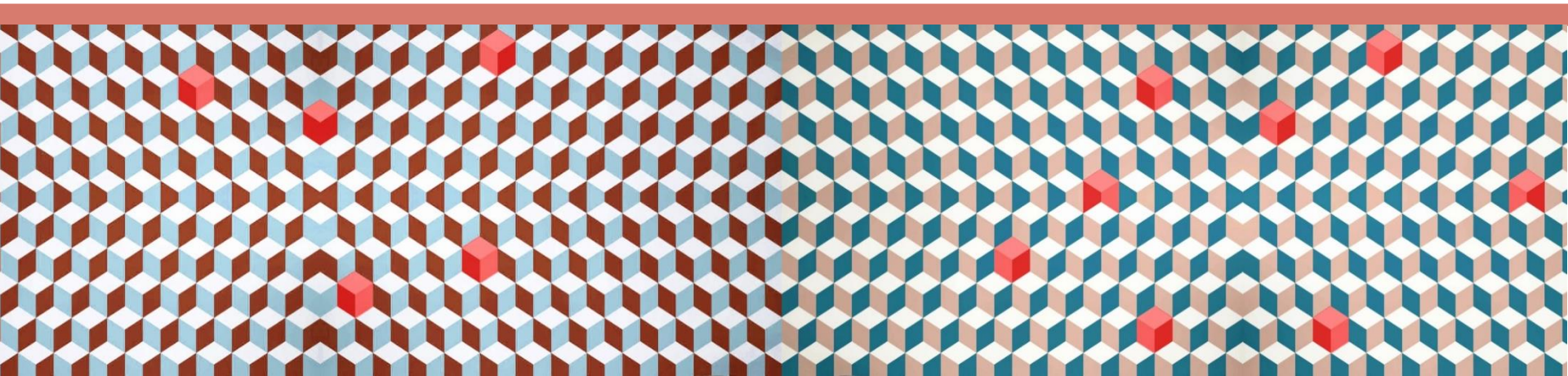
ESERCITAZIONE

Scheduling, Sincronizzazione, Deadlock

Tutor: Giovanni Hauber



Formule

- **TEMPO DI ATTESA**
  - **Normale**: ($T_{completamento} - T_{arrivo} - T_{servizio}$)
  - **Medio**: $\sum(T_{completamento} - T_{arrivo} - T_{servizio}) / N_{processi}$
- **TURNAROUND**
  - **Normale:** $T_{completamento} - T_{arrivo}$
  - **Medio:** $\sum (T_{completamento} - T_{arrivo}) / N_{processi}$
  - **Normalizzato:** $T_{completamento} - T_{arrivo}$ / Burst
- **THROUGHPUT**
  Calcolato come:
  - $N_{processi}$/Tempo Totale

Banchiere

- **Algoritmo del banchiere**: **previene** la formazione dei **deadlock** assicurando che il sistema si trovi in uno **stato tale da garantire il completamento di un processo**, e che quindi le risorse che un processo richiede siano disponibili (**stato sicuro**).
  - **Risorse_Allocate$_{J,K}$**: Quantità di risorse (unità) di una classe R_K **allocate** ad un processo P_J
  - **Risorse_Richieste$_{J,K}$**: Quantità di risorse (unità) di una classe R_K **richieste** dal processo P_J
  - **Risorse_Massime$_{J,K}$**: Quantità di risorse (unità) di una classe R_K richiesta **massime** dal proc. P_J
  - **Risorse_Disponibili$_K$**: Quantità di risorse (unità) di una classe R_K **attualmente nel S.O**.
  - **Risorse_Totali_Allocate:** Quantità di risorse (unità) totali allocate a tutti i pricessi nel sistema: ovvero, $\sum Risorse\_AllocateJK$

- **OBIETTIVO**:

Verificare un plausibile stato sicuro «proiettando» il sistema in uno stato di allocazione futuro, verificando la sequenza di risoluzione ottima dell'esecuzione dei processi.

Banchiere: Risoluzione

1. Definire quali processi si trovano in uno stato sicuro: Per ogni processo, usiamo:
   Risorse_Massime – Risorse_allocate <= Risorse_Disponibili
   • **Se** tutti i processi rispettano ciò, allora il sistema è in uno stato sicuro. Si sceglie quindi il processo che prende più risorse a parità di processi ammissibili.

2. Una volta trovato il processo, si soddisfa la sua rishiesta e una volta terminato, le risorse da esso posseduto vengono rilasciate e sommate alle risorse disponibili:
   Risorse_Diponibili = Risorse_Allocate + Risorse_Disponibili

3. **Torna allo step 1 e ripeti fin quando tutti i processi non sono finiti.**

- Può capite che un processo abbia un numero di risorse allocate incognite. In questo caso, non sappiamo con certezza se il processo sia safe o meno; se le altre risorse ad esso allocate risultano rispettare lo step 1, allora lo consideriamo come tale.
- Quando questo termina e verrà rilasciato inoltre, si risolverà un semplice sistema di disequazioni lineari al fine di stabilire un range di risorse rilasciate da quel processo per quella specifica classe, risolvendo per l'incognita. In forma:

$$\begin{cases} \text{Ris\_massime}_i - \text{Ris\_allocate}_i \geq 0 \\ \text{Ris\_massime}_i - \text{Ris\_allocate}_i \leq \text{Ris\_disponibili}_i \end{cases}$$

Banchiere

	A	B	C	D
P0	4	x-1	3	2
P1	8	0	y-2	2
P2	4	0	0	0
P3	0	0	3	2
P4	2	1	z+1	4

Risorse_Allocate$_{J,K}$

	A	B	C	D
P0	6	4	5	6
P1	10	7	6	8
P2	6	2	0	8
P3	0	3	4	2
P4	9	1	6	9

Risorse_Massime$_{J,K}$

A	B	C	D
2	2	10	4

Risorse_Disponibili

- Determinare gli intervalli sicuri di X,Y,Z per i quali:
  - E' determinata la sequenza sicura del sistema
  - Richiesta di P2[2,0,0,2] è soddisfatta.

Banchiere

A	B	C	D
2	2	10	4

Risorse_Disponibili

- P0: 6-4 ≤ 2? SI | 4-(x-1) ≤ 2? NON SO | 5-3 ≤ 10? SI | 6-2 ≤ 4? SI -> SAFE
- P1: 10-8 ≤ 2? SI | 7-0 <= 2? NO! Mi fermo, processo non sicuro
- P2: 6-4 ≤ 2? SI | 2-0 ≤ 2? SI | 0-0 ≤ 10? SI | 8-0 ≤ 4? NO! . . .
- P3: 0-0 ≤ 2? SI | 3-0 ≤ 2? NO! . . .
- P4: 9-2 ≤ 2? NO! . . .

Quindi di tutti questi scelgo P0! Viene eseguito, termina e rilascia le sue risorse: Essendoci incognita x dobbiamo calcolare il range di risorse rilasciate per la i-sima risorsa dove l'incognita si presenta:

$$\begin{cases} 4-(x-1) \geq 0 \\ 4-(x-1) \leq 2 \end{cases} \longrightarrow \begin{cases} 5 \geq x \\ x \leq 3 \end{cases}$$

RANGE: $3 \leq x \leq 5$

Quindi, rilascio le risorse, sommando quelle allocate a P0 alle risorse disponibili:

$$[2+4,\ 2+(x-1),\ 10+3,\ 4+2] \longrightarrow [6,\ x+1,13,\ 6]$$

Banchiere

	A	B	C	D
	2	2	10	4
P0	6	X+1	13	6

Risorse_Disponibili

- P1: 10-8 ≤ 6? SI | 7-0 ≤ ~~x+1~~ **5+1? (3 ≤ x ≤ 5, e caso peggiore: x=5)?** NO**! . . .**
- P2: 6-4 ≤ 6? SI | 2-0 ≤ x+1? SI | 0-0 ≤ 13? SI | 8-0 ≤ 6? NO! . . .
- P3: 0-0 ≤ 6? SI | 3-0 ≤ x+1? SI | 4-3 ≤ 13? SI | 2-2 ≤ 6? SI -> SAFE
- P4: 9-2 ≤ 6? NO! . . .

Scelgo P3 come processo SAFE, lo eseguo e rilascio le risorse: qui non abbiamo incognite, quindi procedo col semplice rilascio:

[6+0, x-1+0, 13+3, 6+2] ⟶ **[6, x+1,16, 8]**

Banchiere

	A	B	C	D
	2	2	10	4
P0	6	X+1	13	6
P3	6	X+1	16	8

Risorse_Disponibili

- P1: 10-8 ≤ 6? SI | 7-0 ≤ ~~x+1~~ **5+1? (3 ≤ x ≤ 5, e caso peggiore: x=5)?** NO**! . . .**
- P2: 6-4 ≤ 6? SI | 2-0 ≤ x+1? SI | 0-0 ≤ 13? SI | 8-0 ≤ 8? SI -> SAFE
- P4: 9-2 ≤ 6? NO! . . .

Scelgo P2 come processo SAFE, lo eseguo e rilascio le risorse: qui non abbiamo incognite, quindi procedo col semplice rilascio:

[6+4, x-1+0, 16+0, 8+0] ⟶ **[10, x+1,16, 8]**

Banchiere

	A	B	C	D
	2	2	10	4
P0	6	X+1	13	6
P3	6	X+1	16	8
P2	10	X+1	16	8

Risorse_Disponibili

- P1: 10-8 ≤ 6? SI | 7-0 ≤ ~~x+1~~ **5+1? (3 ≤ x ≤ 5, e caso peggiore: x=5)?** NO**! . . .**
- P4: 9-2 ≤ 10? SI | 1-1 ≤ x+1? SI | 6-(z+1) ≤ 16? NON SO | 9-4 ≤ 8? SI -> SAFE

Scelgo P4 come processo SAFE, lo eseguo e rilascio le risorse: essendoci incognita z dobbiamo calcolare il range di risorse rilasciate per la i-sima risorsa dove l'incognita si presenta:

$$\begin{cases} 6-(z+1) \geq 0 \\ 6-(z+1) \leq 16 \end{cases} \longrightarrow \begin{cases} 5 \geq z \\ z \geq -11 \end{cases}$$

RANGE: -11 ≤ z ≤ 5

Quindi, rilascio le risorse, sommando quelle allocate a P4 alle risorse disponibili:

[10+2, x+1+1, 16+z+1, 8+4] ⟶ [12, x+2, z+17, 12]

Banchiere

	A	B	C	D
	2	2	10	4
P0	6	X+1	13	6
P3	6	X+1	16	8
P2	10	X+1	16	8
P4	12	X+2	Z+17	12

Risorse_Disponibili

- P1: 10-8 ≤ 6? SI | 7-0 ≤ x+2? SI | 6 ≤ z+17? NON SO | 8 ≤ 12? SI -> SAFE

Scelgo P1 come processo SAFE, lo eseguo e rilascio le risorse: essendoci incognita y dobbiamo calcolare il range di risorse rilasciate per la i-sima risorsa dove l'incognita si presenta questa volta con una doppia incognita (y,z):

$$\begin{cases} 6-(y-2) \geq 0 \\ 6-(y-2) \leq z+17 \end{cases} \rightarrow$$

Sostituisco z con il suo valore peggiore di allocazione: 5

$$\rightarrow \begin{cases} 8 \geq y \\ -y \leq 5+17-8 \end{cases} \rightarrow \begin{cases} 8 \geq y \\ y \geq -14 \end{cases}$$

RANGE:
$-14 \leq y \leq 8$

Quindi, rilascio le risorse, sommando quelle allocate a P1 alle risorse disponibili:

$$[12+8,\ x+2+0,\ z+17+(y-2),\ 12+2] \longrightarrow [20,\ x+2,\ z+y+15,\ 14]$$

Banchiere

	A	B	C	D
	2	2	10	4
P0	6	X+1	13	6
P3	6	X+1	16	8
P2	10	X+1	16	8
P4	12	X+2	Z+17	12
P1	20	X+2	Z+Y+15	14

Risorse_Disponibili

- La sequenza sicura è: <P0,P3,P2,P4,P1>

- Range X,Y,Z:
  X: $3 \leq x \leq 5$
  Z: $-11 \leq z \leq 5$
  Y: $-14 \leq y \leq 8$

- La richiesta P2[4,0,0,2] Può essere soddisfatta dopo che P0 ha rilasciato le sue risorse.

Scheduling

Esercizio 1

- Si considerino i seguenti processi, attivi in un sistema multiprogrammato:

Processo	Tempo di Arrivo	CPU Burst	Priorità
P1	2ms	14ms	3
P2	8ms	10ms	4
P3	16ms	16ms	5
P4	20ms	12ms	2
P5	20ms	16ms	1

- Supponendo che il cambio di contesto sia 1ms, si mostri l'ordine di esecuzione dei processi e quanto vale il tempo di attesa medio, il tempo di turnaround medio ed il tempo di turnaround normalizzato medio per ciascuno dei seguenti algoritmi di scheduling della CPU:
  - Priorità con prelazione (la priorità massima è 5)
  - Round Robin con quanto q = 4ms

Scheduling

Esercizio 1

- Si considerino i seguenti processi, attivi in un sistema multiprogrammato:

Processo	Tempo di Arrivo	CPU Burst	Priorità
P1	4ms	4ms	3
P2	2ms	10ms	3
P3	4ms	10ms	1
P4	6ms	3ms	2
P5	7ms	6ms	4

- Supponendo che il cambio di contesto sia 3ms, si mostri l'ordine di esecuzione dei processi e quanto vale il tempo di attesa medio, il tempo di turnaround medio ed il tempo di turnaround normalizzato medio per ciascuno dei seguenti algoritmi di scheduling della CPU:
  - Priorità con prelazione (la priorità massima è 1)
  - Round Robin con quanto q = 3ms

Sincronizzazione

- In un ristorante self-service, i clienti, dopo aver mangiato, dispongono i vassoi in M contenitori, ognuno di K ripiani.
- **Periodicamente**, un addetto sceglie un contenitore **tra quelli in cui ci sono più ripiani liberi**, lo svuota, lava i piatti e riporta il contenitore in sala.

Soluzione: Cliente

```
int contenitori [K][M]; // K è il numero di ripiani , M è il
numero di contenitori (colonne=contenitori, righe=ripiani)
semaforo contatore: ripiano_pieno[M]=K;
semaforo contatore: contenitore_disponibile=M;
mutex:accesso_contenitore=1;
indici: contenitore=0;
int ripiani_contenitori[3]=0
int VASSOIO=1
```

```
cliente(){
  while(1) {
    wait(contenitore_disponibile);

    wait(accesso_contenitore)
    contenitore_loc = contenitore++
    signal(accesso_contenitore)

    wait(ripiano_pieno[contenitore_loc])
    contenitori[ripiani_contenitori[contenitore_loc]++][contenitore_loc]=VASSOIO

    wait(accesso_contenitore)
    contenitore=contenitore-1
    signal(accesso_contenitore)

    signal(contenitore_disponibile)
  }
}
```

Scheduling

```
int contenitori [K][M]; // K è il numero di ripiani , M è il
numero di contenitori (questi sarebbero i miei buffer)
semaforo contatore: ripiano_pieno[M]=K;
semaforo contatore: contenitore_disponibile=3;
mutex:accesso_risorse, accesso_contenitore=1;
indici: contenitore=0;
int ripiani_contenitori[M]=0
```

```
addetto() {
  sleep(1)
  min = inf
  ind = -1
  for(i=0;i<M;i++){
    if (ripiani_contenitori[i] <= min) {
    ind = i
    min = ripiani_contenitori[i]
   }
  }

  for(i=0;i<min;i++){
   signal(ripiano_pieno[ind])
   ripiani_contenitore[ind]=0
}
```