

# Sincronizzazione 2 Algoritmi mutua esclusione

martedì 25 novembre 2025 10:01

Vediamoli in dettaglio : data una sezione critica, aggiungo **un preambolo** ( il thread si blocca se non è possibile eseguire sezione critica da parte di quel thread) **ed una coda** (serve per lo sblocco di altri thread , magari notificando qualcuno per farlo eseguire istruzioni). In dettaglio:

## 1. Dekker

<pre>var turno: int;</pre> <p><b>Processo X</b></p> <pre>While turno ≠ X do no-op; &lt;sezione critica&gt;; turno := Y;</pre>	<p><b>Processo Y</b></p> <pre>While turno ≠ Y do no-op; &lt;sezione critica&gt;; turno := X;</pre>
---	--

i. I processi vanno in alternanza stretta nella sezione critica  
• non c'è garanzia di progresso  
• la velocità di esecuzione è limitata dal processo più lento

I processi lavorano come coroutine (ovvero routine che si passano mutuamente il controllo), classiche della strutturazione di un singolo processo, ma inadeguate a processi concorrenti

## 2. Secondo tentativo

<pre>var flag: array[1,n] of boolean;</pre> <p><b>Processo X</b></p> <pre>While flag[Y] do no-op; flag[X] := TRUE; &lt;sezione critica&gt;; flag[X] := FALSE;</pre>	<p><b>Processo Y</b></p> <pre>While flag[X] do no-op; flag[Y] := TRUE; &lt;sezione critica&gt;; flag[Y] := FALSE;</pre>
---	---

i. I processi non vanno in alternanza stretta nella sezione critica  
• c'è garanzia di progresso  
• non c'è garanzia di mutua esclusione (problema che diviene evidente nel caso di numero elevato di processi)

## 3. Terzo tentativo

<pre>var flag: array[1,n] of boolean;</pre> <p><b>Processo X</b></p> <pre>flag[X] := TRUE; While flag[Y] do no-op; &lt;sezione critica&gt;; flag[X] := FALSE;</pre>	<p><b>Processo Y</b></p> <pre>flag[Y] := TRUE; While flag[X] do no-op; &lt;sezione critica&gt;; flag[Y] := FALSE;</pre>
---	---

i.

Possibilità di deadlock, non c'è garanzia di attesa limitata



## 4. Quarto tentativo

<pre>var flag: array[1,n] of boolean;</pre> <p><b>Processo X</b></p> <pre>flag[X] := TRUE; While flag[Y] do {     flag[X] := FALSE;     &lt;pausa&gt;;     flag[X] := TRUE; } &lt;sezione critica&gt;; flag[X] := FALSE;</pre>	<p><b>Processo Y</b></p> <pre>flag[Y] := TRUE; While flag[X] do {     flag[Y] := FALSE;     &lt;pausa&gt;;     flag[Y] := TRUE; } &lt;sezione critica&gt;; flag[Y] := FALSE;</pre>
--	--

Possibilità di starvation, non c'è garanzia di attesa limitata

## 5. Algoritmo (fornoio) - 1974

Basato su assegnazione di numeri per prenotare un turno di accesso alla sezione critica

```
var choosing: array[1,n] of boolean;
number: array[1,n] of int;
repeat {
    choosing[i] := TRUE;
    number[i] := <max in array number[] + 1>;
    choosing[i] := FALSE;
    for j = 1 to n do {
        while choosing[j] do no-op;
        while number[j] ≠ 0 and (number[j],j) < (number[i],i) do no-op;
    }
    <sezione critica>;
    number[i] := 0;
} until FALSE
```

a.

- b. Algoritmo tra repeat e until è algoritmo di sincronizzazione
- c. Per assegnare il numero un thread preleva contatore ed uno lo scrive, ma attenzione potrebbe capitare che venga scritto/letto lo stesso numero.
- d. Comunque venga eseguito si ha busy waiting