

Alcides Calsavara

Relembrando barreira simples



...

Barreira simples:

Base para solução

`n := número total de threads`

`contador := 0 // threads que já executaram o rendezvous`

`mutex := Semáforo(1) // exclusão mútua no contador`

`barreira := Semáforo(0) // bloqueada até que as n
// threads cheguem ao ponto crítico`

Barreira simples:

Solução #2

rendevouz

```
mutex.esperar( )
```

```
    contador := contador + 1
```

```
mutex.sinalizar( )
```

```
se contador == n então barreira.sinalizar( )
```

```
barreira.esperar( )
```

```
barreira.sinalizar( )
```

ponto crítico

Barreira simples



✦ Qual o valor final da barreira na Solução #2 ?

Barreira simples



- ✦ Qual o valor final da barreira na Solução #2 ? 1 (ou mais).

Barreira simples



- ✦ Qual o valor final da barreira na Solução #2 ? 1 (ou mais).
- ✦ O que ocorre com a thread $n+1$? Para ou passa na barreira?

Barreira simples



- ✦ Qual o valor final da barreira na Solução #2 ? **1 (ou mais)**.
- ✦ O que ocorre com a thread $n+1$? Para ou passa na barreira? **PASSA!**

Barreira simples



- ✦ Qual o valor final da barreira na Solução #2 ? **1 (ou mais)**.
- ✦ O que ocorre com a thread $n+1$? Para ou passa na barreira? **PASSA!**
- ✦ Como a thread $n+1$ passa, o que ocorre com a thread $n+2$?

Barreira simples



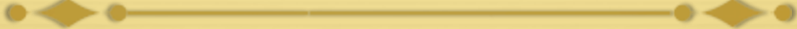
- ✦ Qual o valor final da barreira na Solução #2 ? **1 (ou mais)**.
- ✦ O que ocorre com a thread $n+1$? Para ou passa na barreira? **PASSA!**
- ✦ Como a thread $n+1$ passa, o que ocorre com a thread $n+2$? **TAMBÉM PASSA!**

Barreira simples



- ✦ Qual o valor final da barreira na Solução #2 ? **1 (ou mais)**.
- ✦ O que ocorre com a thread $n+1$? Para ou passa na barreira? **PASSA!**
- ✦ Como a thread $n+1$ passa, o que ocorre com a thread $n+2$? **TAMBÉM PASSA!**
- ✦ Ou seja, depois que as n primeiras threads passarem, todas as demais que chegarem na barreira passarão imediatamente, como se não houvesse barreira.

Barreira reusável



- ✧ Threads em loop: rendezvous – ponto crítico
- ✧ Como fazer para a barreira voltar ao estado inicial (bloqueada) depois que as primeiras n threads tiverem entrado no ponto crítico?
- ✧ Ou seja, a barreira precisa ser reiniciada a cada vez que as n threads executam o ponto crítico.
- ✧ Assim, a barreira volta a permitir a passagem de threads quando as n threads estiverem, novamente, esperando na barreira.

Barreira reusável – Solução #1

rendevouz

```
mutex.esperar( )  
    contador := contador + 1  
mutex.sinalizar( )  
se contador == n então barreira.sinalizar( )
```

```
barreira.esperar( )  
barreira.sinalizar( )
```

ponto crítico

```
mutex.esperar( )  
    contador := contador - 1  
mutex.sinalizar( )  
se contador == 0 então barreira.esperar( )
```


Barreira reusável – Solução #1

rendevouz

```
mutex.esperar( )  
    contador := contador + 1  
mutex.sinalizar( )  
se contador == n então barreira.sinalizar( )
```

Mais de uma thread
pode sinalizar!

```
barreira.esperar( )  
barreira.sinalizar( )
```

ponto crítico

```
mutex.esperar( )  
    contador := contador - 1  
mutex.sinalizar( )  
se contador == 0 então barreira.esperar( )
```

Mais de uma thread
pode esperar!

Barreira reusável – Solução #2

rendevouz

```
mutex.esperar( )  
    contador := contador + 1  
    se contador == n então barreira.sinalizar( )  
mutex.sinalizar( )
```

```
barreira.esperar( )  
barreira.sinalizar( )
```

ponto crítico

```
mutex.esperar( )  
    contador := contador - 1  
    se contador == 0 então barreira.esperar( )  
mutex.sinalizar( )
```

Barreira reusável – Solução #2

rendevouz

```
mutex.esperar( )  
    contador := contador + 1  
    se contador == n então barreira.sinalizar( )  
mutex.sinalizar( )
```

Somente uma thread
pode sinalizar!

```
barreira.esperar( )  
barreira.sinalizar( )
```

ponto crítico

```
mutex.esperar( )  
    contador := contador - 1  
    se contador == 0 então barreira.esperar( )  
mutex.sinalizar( )
```

Somente uma thread
pode esperar!

Barreira reusável – Solução #2

rendevouz

```
mutex.esperar( )  
    contador := contador + 1  
    se contador == n então barreira.sinalizar( )  
mutex.sinalizar( )
```

Somente uma thread
pode sinalizar!

```
barreira.esperar( )  
barreira.sinalizar( )
```

ponto crítico

```
mutex.esperar( )  
    contador := contador - 1  
    se contador == 0 então barreira.esperar( )  
mutex.sinalizar( )
```

Somente uma thread
pode esperar!

Melhorou, mas não evita que uma thread mais rápida ultrapasse as demais!

Barreira reusável – Solução #3

BASE PARA A SOLUÇÃO:

```
barreiraEntrada = Semáforo( 0 ) // fechada
```

```
barreiraSaída = Semáforo( 1 ) // aberta
```

Quando todas as threads chegam na barreiraEntrada, a barreiraSaída é fechada e a barreiraEntrada é aberta.

Quando todas as threads chegam na barreiraSaída, a barreiraEntrada é fechada e a barreiraSaída é aberta.

Barreira reusável – Solução #3

rendevouz

```
mutex.esperar( )
    contador := contador + 1
    se contador == n então
        barreiraSaída.esperar( ) // fecha
        barreiraEntrada.sinalizar( ) // abre
mutex.sinalizar( )
barreiraEntrada.esperar( )
barreiraEntrada.sinalizar( )
ponto crítico
mutex.esperar( )
    contador := contador - 1
    se contador == 0 então
        barreiraEntrada.esperar( ) // fecha
        barreiraSaída.sinalizar( ) // abre
mutex.sinalizar( )
barreiraSaída.esperar( )
barreiraSaída.sinalizar( )
```


Barreira reusável – Solução #3

rendevouz

```
mutex.esperar( )
    contador := contador + 1
    se contador == n então
        barreiraSaída.esperar( ) // fecha
        barreiraEntrada.sinalizar( ) // abre
mutex.sinalizar( )
barreiraEntrada.esperar( )
barreiraEntrada.sinalizar( )
ponto crítico
mutex.esperar( )
    contador := contador - 1
    se contador == 0 então
        barreiraEntrada.esperar( ) // fecha
        barreiraSaída.sinalizar( ) // abre
mutex.sinalizar( )
barreiraSaída.esperar( )
barreiraSaída.sinalizar( )
```

CORRETA

Barreira de duas fases



- ✧ *two-phase barrier*
- ✧ força todas as threads esperarem duas vezes:
 - ✧ primeira vez: até que todas as threads cheguem na entrada da região crítica
 - ✧ segunda vez: até que todas as threads saiam da região crítica