Modelos para Computação Concorrente ou Sistemas Operacionais

Memória Compartilhada – Semáforos – Barreiras

(com slides de Ben-Ari)

Fernando Luís Dotti

PUCRS | ESCOLA POLITÉCNICA

Bibliografia Base

[disponível na biblioteca]

M. Ben-Ari

Principles of Concurrent and Distributed Programming

Second Edition

Addison-Wesley, 2006

Barreiras

Barreiras

Temos vários processos que devem sincronizar em um ponto para depois prosseguirem.
Exemplos:
depois de todos completarem inicialização, pode-se interagir.
depois que todos os processos acabaram, podemos ler todos resultados.

```
P 1 () {
    // computação
    b.arrive()
    // PONTO CRÍTICO
    }
 }
```

```
P 2 () {
    // computação
    b.arrive()
    // PONTO CRÍTICO
    }
 }
```

...

```
P N () {
    // computação
    b.arrive()
    // PONTO CRÍTICO
    }
 }
```

Não confundir ponto crítico com sessão crítica.
Aqui ponto crítico requer que todos processos tenham já chegado ao mesmo para então proceder.

- Barreira

Listing 3.2: Barrier code

```
1 rendezvous
2 critical point
```

Listing 3.3: Barrier hint

```
1 n = the number of threads
2 count = 0
3 mutex = Semaphore(1)
4 barrier = Semaphore(0)
```

?

Barreira

Listing 3.3: Barrier hint

```
1  n = the number of threads
2  count = 0
3  mutex = Semaphore(1)
4  barrier = Semaphore(0)
```

código de "rendezvouz"

Listing 3.5: Barrier solution

```
1  rendezvous
2
3  mutex.wait()
4      count = count + 1
5  mutex.signal()
6
7  if count == n: barrier.signal()
8
9  barrier.wait()
10 barrier.signal()
11
12 critical point
```

n processos executam "rendezvouz"

Listing 3.2: Barrier code

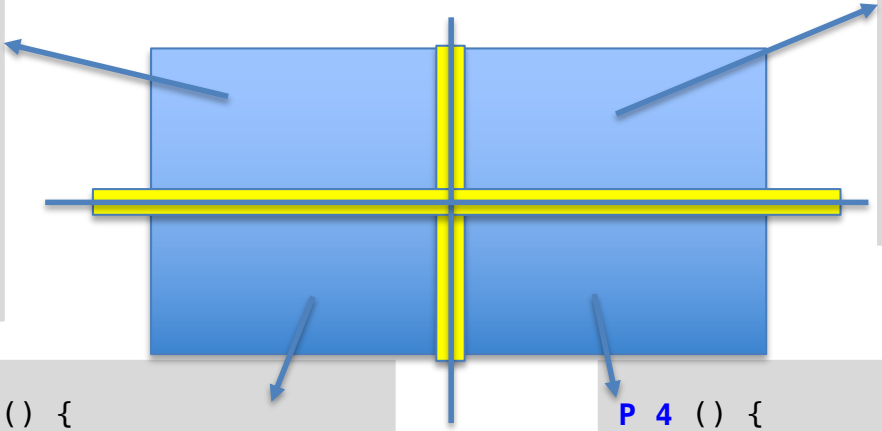
```
1  rendezvous
2  critical point
```

- Barreira "simples"
  - após uso, variáveis ficam em estado que não permite reuso da barreira
  - muitas aplicações necessitam de rodadas de sincronização
  - barreiras "reutilizáveis"

Barreiras Reutilizáveis

Exemplo: uma imagem é dividida em partes para ser processada em paralelo por diferentes processos. O processamento é em fases. Em cada fase, cada processo deve ler o valor da sua parte, e os pixels da borda das vizinhas, calcular os novos valores, atribuir, partir para proxima fase. Todos processos devem ler valores da mesma fase, em todas partições vizinhas para calclular seu valor na próxima fase. Ou seja, se um processo qualquer está ainda lendo, nenhum pode escrever, se algum está escrevendo, nenhum pode ler.

```
P 1 () {
  loop
    {
    // PONTO NÃO CRÍTICO
    // Lê valores, calcula
    b.arrive()
    // PONTO CRÍTICO
    // Escreve
    b.leave()
    }
 }
```



```
P 2 () {
  loop
    {
    // PONTO NÃO CRÍTICO
    b.arrive()
    // PONTO CRÍTICO
    b.leave()
    }
 }
```

```
P 3 () {
  loop
    {
    // PONTO NÃO CRÍTICO

    b.arrive()
    // PONTO CRÍTICO

    b.leave()
    }
 }
```

```
P 4 () {
  loop
    {
    // PONTO NÃO CRÍTICO

    b.arrive()
    // PONTO CRÍTICO

    b.leave()
    }
 }
```

- Barreira reutilizável
  - adiciona fase de sincronização na saída do ponto crítico

Listing 3.9: Reusable barrier hint

```
1 turnstile = Semaphore(0)
2 turnstile2 = Semaphore(1)
3 mutex = Semaphore(1)
```

- Barreira reutilizável

Listing 3.9: Reusable barrier hint

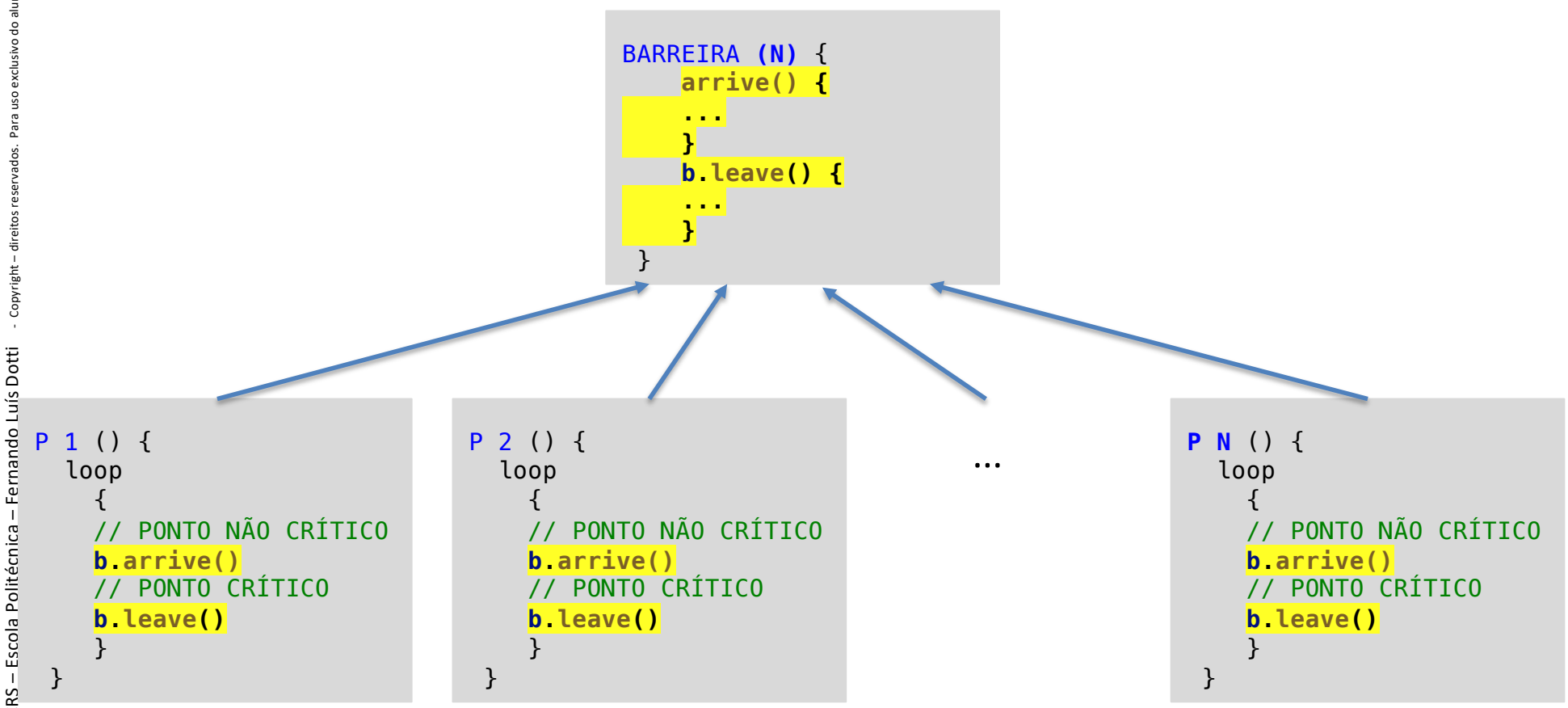
```
1  turnstile = Semaphore(0)
2  turnstile2 = Semaphore(1)
3  mutex = Semaphore(1)
```

Listing 3.10: Reusable barrier solution

```
1  # rendezvous
2
3  mutex.wait()
4      count += 1
5      if count == n:
6          turnstile2.wait()     # lock the second
7          turnstile.signal()    # unlock the first
8  mutex.signal()
9
10 turnstile.wait()              # first turnstile
11 turnstile.signal()
12
13 # critical point
14
15 mutex.wait()
16     count -= 1
17     if count == 0:
18         turnstile.wait()      # lock the first
19         turnstile2.signal()   # unlock the second
20 mutex.signal()
21
22 turnstile2.wait()             # second turnstile
23 turnstile2.signal()
```

Barreiras Reutilizáveis

- processos devem sincronizar (fazer encontro temporal) antes de iniciar uma outra fase



```java
class Barrier {

    private Semaphore mutex;
    private Semaphore catraca1;
    private Semaphore catraca2;
    private int count; // nro que chegou
    private int n; // nro de threads a chegar na barreira

    public Barrier(int nmax){
        n = nmax;                          → n = 3
        count = 0;                         → count = 0
        mutex = new Semaphore(1);          → mutex = (1,{})
        catraca1 = new Semaphore(0);       → catraca1 = (0,{})
        catraca2 = new Semaphore(1);       → catraca2 = (1,{})
    }

    public void arrive() throws InterruptedException {
        mutex.acquire();
        count++;
        if (count == n) {
            catraca2.acquire();
            catraca1.release();
        }
        mutex.release();
        catraca1.acquire();
        catraca1.release();
    }

    public void leave() throws InterruptedException {
        mutex.acquire();
        count--;
        if (count == 0) {
            catraca1.acquire();
            catraca2.release();
        }
        mutex.release();
        catraca2.acquire();
        catraca2.release();
    }
}
```

suponha threads t1, t2, t3,
com barreira = 3
fazem arrive e depois leave, em loop

t1	t2	t3	ponto não crítico

situação inicial

```
class Barrier {

    private Semaphore mutex;
    private Semaphore catraca1;
    private Semaphore catraca2;
    private int count; // nro que chegou
    private int n; // nro de threads a chegar na barreira

    public Barrier(int nmax){
        n = nmax;                          → n = 3
        count = 0;                         → count = 0
        mutex = new Semaphore(1);          → mutex = (0,{})
        catraca1 = new Semaphore(0);       → catraca1 = (0,{})
        catraca2 = new Semaphore(1);       → catraca2 = (1,{})
    }

    public void arrive() throws InterruptedException {
        mutex.acquire();        <- t1 entrou
        count++;
        if (count == n) {
            catraca2.acquire();
            catraca1.release();
        }
        mutex.release();
        catraca1.acquire();
        catraca1.release();
    }

    public void leave() throws InterruptedException {
        mutex.acquire();
        count--;
        if (count == 0) {
            catraca1.acquire();
            catraca2.release();
        }
        mutex.release();
        catraca2.acquire();
        catraca2.release();
    }
}
```

t2 t3 ponto não crítico

```
class Barrier {

    private Semaphore mutex;
    private Semaphore catraca1;
    private Semaphore catraca2;
    private int count; // nro que chegou
    private int n; // nro de threads a chegar na barreira

    public Barrier(int nmax){
        n = nmax;                          → n = 3
        count = 0;                         → count = 1
        mutex = new Semaphore(1);          → mutex = (0,{ t2, t3 })
        catraca1 = new Semaphore(0);       → catraca1 = (0,{})
        catraca2 = new Semaphore(1);       → catraca2 = (1,{})
    }

    public void arrive() throws InterruptedException {
        mutex.acquire();        <- t2      <- t3        t2 e t3 chegam
        count++;                <-  t1                  t1 incr count
        if (count == n) {
            catraca2.acquire();
            catraca1.release();
        }
        mutex.release();
        catraca1.acquire();
        catraca1.release();
    }

    public void leave() throws InterruptedException {
        mutex.acquire();
        count--;
        if (count == 0) {
            catraca1.acquire();
            catraca2.release();
        }
        mutex.release();
        catraca2.acquire();
        catraca2.release();
    }
}
```

```
class Barrier {

    private Semaphore mutex;
    private Semaphore catraca1;
    private Semaphore catraca2;
    private int count; // nro que chegou
    private int n; // nro de threads a chegar na barreira

    public Barrier(int nmax){
        n = nmax;                          → n = 3
        count = 0;                         → count = 1
        mutex = new Semaphore(1);          → mutex = (0,{ t3 })
        catraca1 = new Semaphore(0);       → catraca1 = (0,{ t1 })
        catraca2 = new Semaphore(1);       → catraca2 = (1,{})
    }

    public void arrive() throws InterruptedException {
        mutex.acquire();          <- t2        <- t3
        count++;
        if (count == n) {
            catraca2.acquire();
            catraca1.release();
        }
        mutex.release();
        catraca1.acquire();       <-  t1
        catraca1.release();
    }

    public void leave() throws InterruptedException {
        mutex.acquire();
        count--;
        if (count == 0) {
            catraca1.acquire();
            catraca2.release();
        }
        mutex.release();
        catraca2.acquire();
        catraca2.release();
    }
}
```

t1 nao entra no if,
faz mutex release, acordando t2, e vai para catraca1

```
class Barrier {

    private Semaphore mutex;
    private Semaphore catraca1;
    private Semaphore catraca2;
    private int count; // nro que chegou
    private int n; // nro de threads a chegar na barreira

    public Barrier(int nmax){
        n = nmax;                          → n = 3
        count = 0;                         → count = 2
        mutex = new Semaphore(1);          → mutex = (0,{})
        catraca1 = new Semaphore(0);       → catraca1 = (0,{ t1, t2 })
        catraca2 = new Semaphore(1);       → catraca2 = (1,{})
    }

    public void arrive() throws InterruptedException {
        mutex.acquire();
        count++;
        if (count == n) {
            catraca2.acquire();
            catraca1.release();
        }
        mutex.release();
        catraca1.acquire();
        catraca1.release();
    }

    public void leave() throws InterruptedException {
        mutex.acquire();
        count--;
        if (count == 0) {
            catraca1.acquire();
            catraca2.release();
        }
        mutex.release();
        catraca2.acquire();
        catraca2.release();
    }
}
```

<- t3

<- t2

t2 prossegue, incr count,
solta mutex, que libera t3 , e vai p catraca1

<- t2

<- t1 <- t2

```
class Barrier {

    private Semaphore mutex;
    private Semaphore catraca1;
    private Semaphore catraca2;
    private int count; // nro que chegou
    private int n; // nro de threads a chegar na barreira

    public Barrier(int nmax){
        n = nmax;                          → n = 3
        count = 0;                         → count = 3
        mutex = new Semaphore(1);          → mutex = (0,{})
        catraca1 = new Semaphore(0);       → catraca1 = (0,{ t2 })
        catraca2 = new Semaphore(1);       → catraca2 = (0,{})
    }

    public void arrive() throws InterruptedException {
        mutex.acquire();
        count++;                    <- t3
        if (count == n) {           <- t3
            catraca2.acquire();     <- t3
            catraca1.release();     <- t3
        }
        mutex.release();
        catraca1.acquire();  <- t1  <- t2
        catraca1.release();
    }

    public void leave() throws InterruptedException {
        mutex.acquire();
        count--;
        if (count == 0) {
            catraca1.acquire();
            catraca2.release();
        }
        mutex.release();
        catraca2.acquire();
        catraca2.release();
    }
}
```

t3 prossegue, incr count,
entra no if, aquire catraca 2,
release catraca 1 (desbloq t1)

```
class Barrier {

    private Semaphore mutex;
    private Semaphore catraca1;
    private Semaphore catraca2;
    private int count; // nro que chegou
    private int n; // nro de threads a chegar na barreira

    public Barrier(int nmax){
        n = nmax;                          → n = 3
        count = 0;                         → count = 3
        mutex = new Semaphore(1);          → mutex = (1,{})
        catraca1 = new Semaphore(0);       → catraca1 = (0,{ t2, t3 })
        catraca2 = new Semaphore(1);       → catraca2 = (0,{})
    }

    public void arrive() throws InterruptedException {
        mutex.acquire();
        count++;
        if (count == n) {
            catraca2.acquire();
            catraca1.release();
        }
        mutex.release();
        catraca1.acquire();          <- t2 <- t3
        catraca1.release();  <-  t1
    }

    public void leave() throws InterruptedException {
        mutex.acquire();
        count--;
        if (count == 0) {
            catraca1.acquire();
            catraca2.release();
        }
        mutex.release();
        catraca2.acquire();
        catraca2.release();
    }
}
```

t3 solta mutex, entra na catraca1,
t1 vai para o release

```
class Barrier {

    private Semaphore mutex;
    private Semaphore catraca1;
    private Semaphore catraca2;
    private int count; // nro que chegou
    private int n; // nro de threads a chegar na barreira

    public Barrier(int nmax){
        n = nmax;                          → n = 3
        count = 0;                         → count = 3
        mutex = new Semaphore(1);          → mutex = (1,{})
        catraca1 = new Semaphore(0);       → catraca1 = (0,{ t3 })
        catraca2 = new Semaphore(1);       → catraca2 = (0,{})
    }

    public void arrive() throws InterruptedException {
        mutex.acquire();
        count++;
        if (count == n) {
            catraca2.acquire();
            catraca1.release();
        }
        mutex.release();
        catraca1.acquire();
        catraca1.release();
    }

    public void leave() throws InterruptedException {
        mutex.acquire();
        count--;
        if (count == 0) {
            catraca1.acquire();
            catraca2.release();
        }
        mutex.release();
        catraca2.acquire();
        catraca2.release();
    }
}
```

t1 release catraca 1 e sai do arrive

t2 sai da catraca 1

<- t2

<- t3

t1 ponto crítico

```
class Barrier {

    private Semaphore mutex;
    private Semaphore catraca1;
    private Semaphore catraca2;
    private int count; // nro que chegou
    private int n; // nro de threads a chegar na barreira

    public Barrier(int nmax){
        n = nmax;                          → n = 3
        count = 0;                         → count = 3
        mutex = new Semaphore(1);          → mutex = (1,{})
        catraca1 = new Semaphore(0);       → catraca1 = (0,{})
        catraca2 = new Semaphore(1);       → catraca2 = (0,{})
    }

    public void arrive() throws InterruptedException {
        mutex.acquire();
        count++;
        if (count == n) {
            catraca2.acquire();
            catraca1.release();
        }
        mutex.release();
        catraca1.acquire();
        catraca1.release();     <- t3
    }

    public void leave() throws InterruptedException {
        mutex.acquire();
        count--;
        if (count == 0) {
            catraca1.acquire();
            catraca2.release();
        }
        mutex.release();
        catraca2.acquire();
        catraca2.release();
    }
}
```

t2 release catraca 1 e sai do arrive

t2 sai da catraca 1

t1	t2	ponto crítico

```
class Barrier {

    private Semaphore mutex;
    private Semaphore catraca1;
    private Semaphore catraca2;
    private int count; // nro que chegou
    private int n; // nro de threads a chegar na barreira

    public Barrier(int nmax){
        n = nmax;                          → n = 3
        count = 0;                         → count = 3
        mutex = new Semaphore(1);          → mutex = (1,{})
        catraca1 = new Semaphore(0);       → catraca1 = (1,{})
        catraca2 = new Semaphore(1);       → catraca2 = (0,{})
    }

    public void arrive() throws InterruptedException {
        mutex.acquire();
        count++;
        if (count == n) {
            catraca2.acquire();
            catraca1.release();
        }
        mutex.release();
        catraca1.acquire();
        catraca1.release();
    }

    public void leave() throws InterruptedException {
        mutex.acquire();
        count--;
        if (count == 0) {
            catraca1.acquire();
            catraca2.release();
        }
        mutex.release();
        catraca2.acquire();
        catraca2.release();
    }
}
```

t3 release catraca 1 e sai do arrive
catraca 1 fica em 1, pronta para o leave

t1	t2	t3	ponto crítico

```
class Barrier {

    private Semaphore mutex;
    private Semaphore catraca1;
    private Semaphore catraca2;
    private int count; // nro que chegou
    private int n; // nro de threads a chegar na barreira

    public Barrier(int nmax){
        n = nmax;                          → n = 3
        count = 0;                         → count = 3
        mutex = new Semaphore(1);          → mutex = (1,{})
        catraca1 = new Semaphore(0);       → catraca1 = (1,{})
        catraca2 = new Semaphore(1);       → catraca2 = (0,{})
    }

    public void arrive() throws InterruptedException {
        mutex.acquire();
        count++;
        if (count == n) {
            catraca2.acquire();
            catraca1.release();
        }
        mutex.release();
        catraca1.acquire();
        catraca1.release();
    }

    public void leave() throws InterruptedException {
        mutex.acquire();
        count--;
        if (count == 0) {
            catraca1.acquire();
            catraca2.release();
        }
        mutex.release();
        catraca2.acquire();
        catraca2.release();
    }
}
```

vamos supor mesma ordem de chegada no leave, por simplicidade

<- t1 t2 t3 ponto crítico

```
class Barrier {

    private Semaphore mutex;
    private Semaphore catraca1;
    private Semaphore catraca2;
    private int count; // nro que chegou
    private int n; // nro de threads a chegar na barreira

    public Barrier(int nmax){
        n = nmax;                          → n = 3
        count = 0;                         → count = 2
        mutex = new Semaphore(1);          → mutex = (0,{})
        catraca1 = new Semaphore(0);       → catraca1 = (1,{})
        catraca2 = new Semaphore(1);       → catraca2 = (0,{})
    }

    public void arrive() throws InterruptedException {
        mutex.acquire();
        count++;
        if (count == n) {
            catraca2.acquire();
            catraca1.release();
        }
        mutex.release();
        catraca1.acquire();
        catraca1.release();
    }

    public void leave() throws InterruptedException {
        mutex.acquire();
        count--;                  <- t1
        if (count == 0) {
            catraca1.acquire();
            catraca2.release();
        }
        mutex.release();
        catraca2.acquire();
        catraca2.release();
    }
}
```

vamos supor mesma ordem de chegada no leave, por simplicidade

t2 t3 ponto crítico

t1 pega mutex, decr count

```
class Barrier {

    private Semaphore mutex;
    private Semaphore catraca1;
    private Semaphore catraca2;
    private int count; // nro que chegou
    private int n; // nro de threads a chegar na barreira

    public Barrier(int nmax){
        n = nmax;                          → n = 3
        count = 0;                         → count = 2
        mutex = new Semaphore(1);          → mutex = (0,{ t3 })
        catraca1 = new Semaphore(0);       → catraca1 = (1,{})
        catraca2 = new Semaphore(1);       → catraca2 = (0,{})
    }

    public void arrive() throws InterruptedException {
        mutex.acquire();
        count++;
        if (count == n) {
            catraca2.acquire();
            catraca1.release();
        }
        mutex.release();
        catraca1.acquire();
        catraca1.release();
    }

    public void leave() throws InterruptedException {
        mutex.acquire();
        count--;
        if (count == 0) {
            catraca1.acquire();
            catraca2.release();
        }
        mutex.release();
        catraca2.acquire();
        catraca2.release();
    }
}
```

<- t2 <- t3

<- t1

vamos supor mesma ordem de chegada no leave, por simplicidade

t1 libera mutex
t2 pega o mutex
t3 bloqueia no mutex

```
class Barrier {

    private Semaphore mutex;
    private Semaphore catraca1;
    private Semaphore catraca2;
    private int count; // nro que chegou
    private int n; // nro de threads a chegar na barreira

    public Barrier(int nmax){
        n = nmax;                          → n = 3
        count = 0;                         → count = 1
        mutex = new Semaphore(1);          → mutex = (0,{ })
        catraca1 = new Semaphore(0);       → catraca1 = (1,{})
        catraca2 = new Semaphore(1);       → catraca2 = (0,{ t1 })
    }

    public void arrive() throws InterruptedException {
        mutex.acquire();
        count++;
        if (count == n) {
            catraca2.acquire();
            catraca1.release();
        }
        mutex.release();
        catraca1.acquire();
        catraca1.release();
    }

    public void leave() throws InterruptedException {
        mutex.acquire();
        count--;                    <- t3
        if (count == 0) {
            catraca1.acquire();
            catraca2.release();
        }
        mutex.release();    <- t2
        catraca2.acquire();    <- t1
        catraca2.release();
    }
}
```

vamos supor mesma ordem de chegada no leave, por simplicidade

t1 chega na catraca 2
t2 decr count libera o mutex
t3 desbloqueia e entra

```
class Barrier {

    private Semaphore mutex;
    private Semaphore catraca1;
    private Semaphore catraca2;
    private int count; // nro que chegou
    private int n; // nro de threads a chegar na barreira

    public Barrier(int nmax){
        n = nmax;                          → n = 3
        count = 0;                         → count = 0
        mutex = new Semaphore(1);          → mutex = (0,{ })
        catraca1 = new Semaphore(0);       → catraca1 = (0,{})
        catraca2 = new Semaphore(1);       → catraca2 = (0,{ })
    }

    public void arrive() throws InterruptedException {
        mutex.acquire();
        count++;
        if (count == n) {
            catraca2.acquire();
            catraca1.release();
        }
        mutex.release();
        catraca1.acquire();
        catraca1.release();
    }

    public void leave() throws InterruptedException {
        mutex.acquire();
        count--;
        if (count == 0) {
            catraca1.acquire();
            catraca2.release();
        }
        mutex.release();
        catraca2.acquire();
        catraca2.release();
    }
}
```

<- t3

<- t2

<- t1

vamos supor mesma ordem de chegada no leave, por simplicidade

t3 decr count e entra no if

acquire catraca 1, release catraca 2 (desbloq t1)

!! Note que o desbloqueio da catraca iniciou mesmo que nem todos tenham chegado a ela!!! O entanto o que importa é que todas threads necessárias estão no escopo da operação leave !!

O mesmo vale para arrive.

```java
class Barrier {

    private Semaphore mutex;
    private Semaphore catraca1;
    private Semaphore catraca2;
    private int count; // nro que chegou
    private int n; // nro de threads a chegar na barreira

    public Barrier(int nmax){
        n = nmax;                          → n = 3
        count = 0;                         → count = 0
        mutex = new Semaphore(1);          → mutex = (1,{ })
        catraca1 = new Semaphore(0);       → catraca1 = (0,{})
        catraca2 = new Semaphore(1);       → catraca2 = (0,{ t2, t3 })
    }

    public void arrive() throws InterruptedException {
        mutex.acquire();
        count++;
        if (count == n) {
            catraca2.acquire();
            catraca1.release();
        }
        mutex.release();
        catraca1.acquire();
        catraca1.release();
    }

    public void leave() throws InterruptedException {
        mutex.acquire();
        count--;
        if (count == 0) {
            catraca1.acquire();
            catraca2.release();
        }
        mutex.release();
        catraca2.acquire();
        catraca2.release();
    }
}
```

<- t2 <- t1 <- t3

vamos supor mesma ordem de chegada no leave, por simplicidade

t2 entra na catraca 2
t3 libera mutex, entra na catraca 2
t1 vai para o release

```
class Barrier {

    private Semaphore mutex;
    private Semaphore catraca1;
    private Semaphore catraca2;
    private int count; // nro que chegou
    private int n; // nro de threads a chegar na barreira

    public Barrier(int nmax){
        n = nmax;                          → n = 3
        count = 0;                         → count = 0
        mutex = new Semaphore(1);          → mutex = (1,{ })
        catraca1 = new Semaphore(0);       → catraca1 = (0,{})
        catraca2 = new Semaphore(1);       → catraca2 = (0,{ t3 })
    }

    public void arrive() throws InterruptedException {
        mutex.acquire();
        count++;
        if (count == n) {
            catraca2.acquire();
            catraca1.release();
        }
        mutex.release();
        catraca1.acquire();
        catraca1.release();
    }

    public void leave() throws InterruptedException {
        mutex.acquire();
        count--;
        if (count == 0) {
            catraca1.acquire();
            catraca2.release();
        }
        mutex.release();
        catraca2.acquire();
        catraca2.release();
    }
}
```

<- t2

<- t3

vamos supor mesma ordem de chegada no leave, por simplicidade

t1 release catraca 2, acorda t2, e sai do leave
t2 vai para o release

t1	ponto não crítico

```java
class Barrier {

    private Semaphore mutex;
    private Semaphore catraca1;
    private Semaphore catraca2;
    private int count; // nro que chegou
    private int n; // nro de threads a chegar na barreira

    public Barrier(int nmax){
        n = nmax;                          → n = 3
        count = 0;                         → count = 0
        mutex = new Semaphore(1);          → mutex = (1,{ })
        catraca1 = new Semaphore(0);       → catraca1 = (0,{})
        catraca2 = new Semaphore(1);       → catraca2 = (1,{})
    }

    public void arrive() throws InterruptedException {
        mutex.acquire();
        count++;
        if (count == n) {
            catraca2.acquire();
            catraca1.release();
        }
        mutex.release();
        catraca1.acquire();
        catraca1.release();
    }

    public void leave() throws InterruptedException {
        mutex.acquire();
        count--;
        if (count == 0) {
            catraca1.acquire();
            catraca2.release();
        }
        mutex.release();
        catraca2.acquire();
        catraca2.release();     <- t3
    }
}
```

vamos supor mesma ordem de chegada no leave, por simplicidade

t2 release catraca 2, acorda t3, e sai do leave
t3 vai para o release da catraca 2, deixando em 1
e sai do leave

t1 t2 t3 ponto não crítico

```
class Barrier {

    private Semaphore mutex;
    private Semaphore catraca1;
    private Semaphore catraca2;
    private int count; // nro que chegou
    private int n; // nro de threads a chegar na barreira

    public Barrier(int nmax){
        n = nmax;                          → n = 3
        count = 0;                         → count = 0
        mutex = new Semaphore(1);          → mutex = (1,{ })
        catraca1 = new Semaphore(0);       → catraca1 = (0,{})
        catraca2 = new Semaphore(1);       → catraca2 = (1,{})
    }

    public void arrive() throws InterruptedException {
        mutex.acquire();
        count++;
        if (count == n) {
            catraca2.acquire();
            catraca1.release();
        }
        mutex.release();
        catraca1.acquire();
        catraca1.release();
    }

    public void leave() throws InterruptedException {
        mutex.acquire();
        count--;
        if (count == 0) {
            catraca1.acquire();
            catraca2.release();
        }
        mutex.release();
        catraca2.acquire();
        catraca2.release();
    }
}
```

t1 t2 t3 ponto não crítico

!! note que o estado da barreira depois de todas threads terem entrado e saído
é o estado inicial.
então a barreira pode ser reutilizada!!