Geek University

Evolua seu lado geek!

www.geekuniversity.com.br

Sincronização das Threads

Sincronização das Threads

Quando trabalhamos com multi-threads é recomendável que tentemos manter o mínimo possível de recursos compartilhados entre as threads.

Ou seja, evite que uma thread dependa de dados ou recursos que outra thread possa manipular.

Quanto maior o nível de compartilhamento de recursos/dados/memória através de threads que houver no seu programa, mais complexo será o gerenciamento e resultados inesperados certamente ocorrerão.

Para ajudar a contornar os problemas de interferência em threads (race conditions), existem diversos mecanismos que nos ajudam a controlar o acesso a determinado recurso por uma thread.

Sincronização das Threads

Lock

O principal e mais fundamental mecanismo de controle de threads para acesso à recursos (dados/memória/dispositivos/etc) é o lock.

Sincronização das Threads

Lock

O lock, como o próprio nome sugere, é um recurso de bloqueio. Ou seja, usamos para que a thread bloqueie o acesso a determinado recurso.

Este mecanismo possui dois estados:

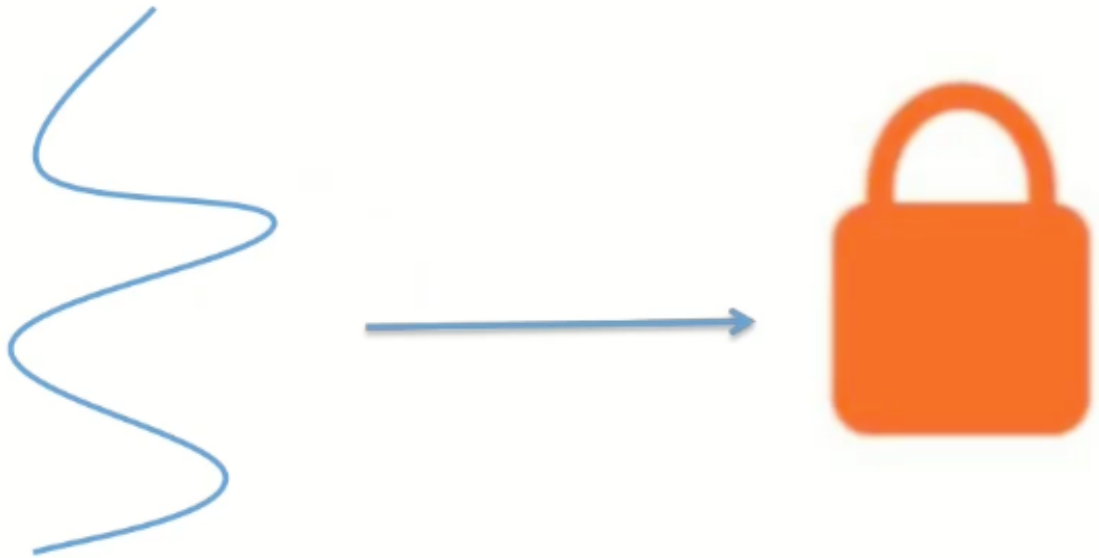
Unlock
(Desbloqueado)

Lock
(Bloqueado)

Sincronização das Threads

Lock

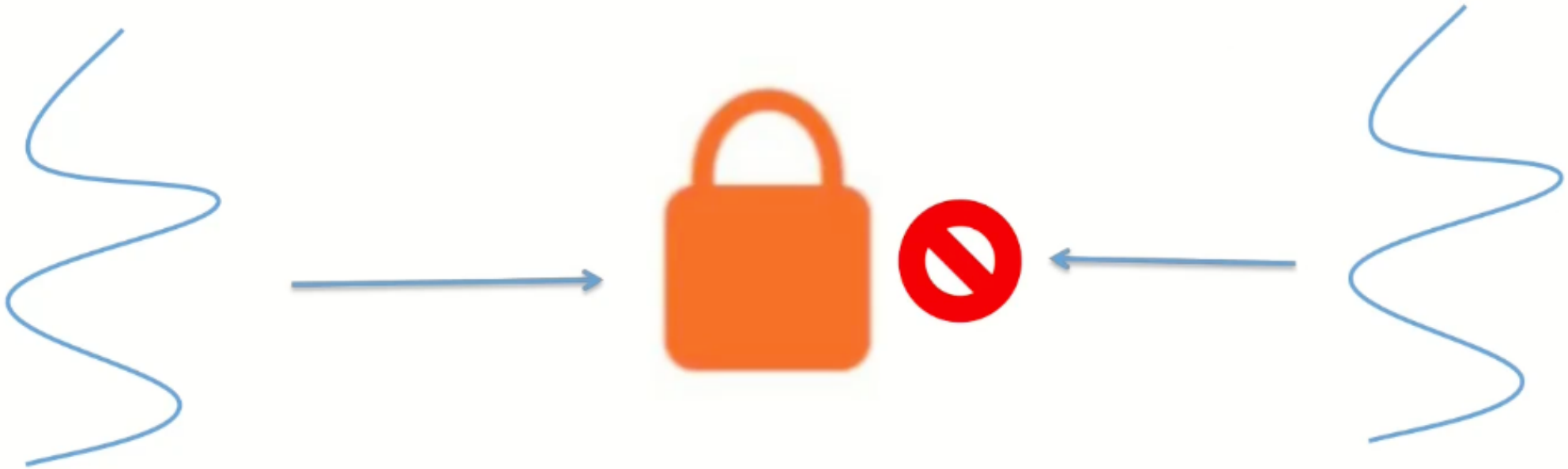
Quando uma thread realiza um lock em um recurso ela faz com que nenhuma outra thread tenha acesso a este recurso.



Sincronização das Threads

Lock

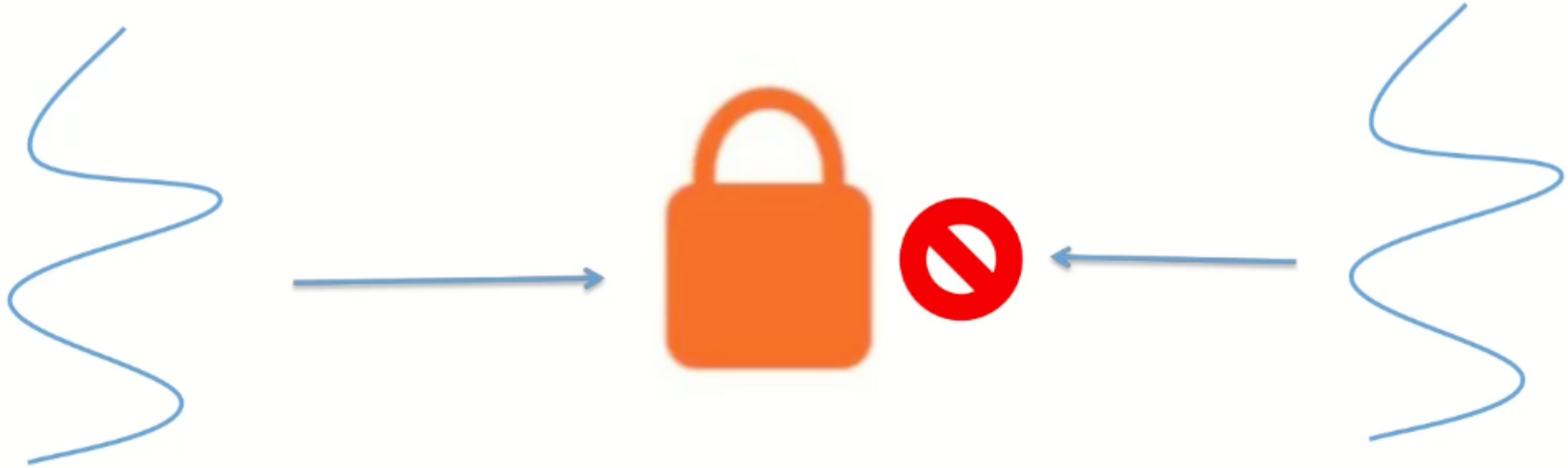
Quando uma thread realiza um lock em um recurso ela faz com que nenhuma outra thread tenha acesso a este recurso. Além disso a única thread que pode realizar um unlock é ela mesma.



Sincronização das Threads

Lock

Quando uma thread realiza um lock em um recurso ela faz com que nenhuma outra thread tenha acesso a este recurso. Além disso a única thread que pode realizar um unlock é ela mesma.



Uma thread adquire um lock com o método *acquire()* e realiza um unlock com o método *release()*

Sincronização das Threads

Lock

Quando uma thread realiza um lock em um recurso ela faz com que nenhuma outra thread tenha acesso a este recurso. Além disso a única thread que pode realizar um unlock é ela mesma.

```python
import threading


th = threading.Thread(target=contar, args=('elefante', 10))


lock = th.Lock()


# Lock
lock.acquire()
# Realiza qualquer operação com o recurso bloqueado para outras threads...

# Unlock
lock.release()

```

Uma thread adquire um lock com o método *acquire()* e realiza um unlock com o método *release()*

Sincronização das Threads

Lock

Um problema que temos aqui é que caso ocorra alguma exceção durante as ações efetuadas com o acquire(), o método release() nunca será executado, e desta forma o recurso continuará bloqueado.

```python
import threading


th = threading.Thread(target=contar, args=('elefante', 10))


lock = th.Lock()


# Lock
lock.acquire()
# Realiza qualquer operação com o recurso bloqueado para outras threads...

# Unlock
lock.release()

```

Sincronização das Threads

Lock

Conseguimos resolver este problema facilmente realizando o tratamento da possível exceção e executando o método *release()* na cláusula *finally*.

```
1  import threading
2
3
4  th = threading.Thread(target=contar, args=('elefante', 10))
5
6
7  lock = th.Lock()
8
9
10 # Lock
11 lock.acquire()
12 try:
13     # Realiza qualquer operação com o recurso bloqueado para outras threads...
14 except:
15     # Trata a exceção...
16 finally:
17     # Unlock
18     lock.release()
19
```

Sincronização das Threads

Lock

Podemos ainda fazer uso dos gerenciadores de contexto, como o with, para que os métodos *acquire()* e *release()* sejam executados automaticamente ao entrar no bloco do contexto e ao sair dele.

```
1  import threading
2
3
4  th = threading.Thread(target=contar, args=('elefante', 10))
5
6
7  lock = th.Lock()
8
9
10 with lock:
11     # Realiza qualquer operação com o recurso bloqueado para outras threads...
12
13
```

Sincronização das Threads

Lock

Lembre-se que se uma thread tentar acessar um recurso que está bloqueado por outra thread não terá acesso e desta forma ficará aguardando o recurso ser liberado para poder trabalhar.

Você pode não querer ficar aguardando...por exemplo poderá fazer outra coisa e depois voltar ao recurso para ver se ele já foi liberado.

Sincronização das Threads

Lock

Lembre-se que se uma thread tentar acessar um recurso que está bloqueado por outra thread não terá acesso e desta forma ficará aguardando o recurso ser liberado para poder trabalhar.

Você pode não querer ficar aguardando...por exemplo poderá fazer outra coisa e depois voltar ao recurso para ver se ele já foi liberado.

```
1  import threading
2
3
4  th = threading.Thread(target=contar, args=('elefante', 10))
5
6
7  lock = th.Lock()
8
9
10 if lock.acquire(False):
11     # Posso bloquear o recurso e utilizá-lo...
12 else:
13     # O recurso já está bloqueado por outra thread...
14     # vou fazer outras coisas e tento novamente mais tarde
15
16
```

Sincronização das Threads

Lock

Imaginando que seu código tem um loop e dentro deste loop você terá threads sendo executadas...tome cuidado ao utilizar o Lock...

```
1  import threading
2
3
4  th = threading.Thread(target=contar, args=('elefante', 10))
5
6
7  lock = th.Lock()
8
9  lock.acquire()
10 lock.acquire() # Esta execução irá bloquear a thread
11
```

Ou melhor dizendo...nunca use o Lock...

Sincronização das Threads

Lock

Utilize sempre o RLock que faz as mesmas coisas que o Lock mas não bloqueia sua própria thread em caso de tentativa de adquirir lock em um recurso que já está bloqueado pela própria thread.

```
1  import threading
2
3
4  th = threading.Thread(target=contar, args=('elefante', 10))
5
6
7  lock = th.RLock()
8
9  lock.acquire()
10 lock.acquire() # Esta execução não irá bloquear a thread
11
```

Sincronização das Threads

Lock

Utilize sempre o RLock que faz as mesmas coisas que o Lock mas não bloqueia sua própria thread em caso de tentativa de adquirir lock em um recurso que já está bloqueado pela própria thread.

```
1  import threading
2
3
4  th = threading.Thread(target=contar, args=('elefante', 10))
5
6
7  lock = th.RLock()
8
9  lock.acquire()
10 lock.acquire() # Esta execução não irá bloquear a thread
11
```

Na próxima aula iremos falar um pouco sobre comunicação entre threads…

Geek University

Evolua seu lado geek!

www.geekuniversity.com.br