Semaphores

Ryan Eberhardt
July 26, 2021
Semaphores

thread1
Semaphores

If necessary, waits for a ball to be added to the bucket; then, takes the ball.
Semaphores

```java
semaphore.wait()
```

If necessary, waits for a ball to be added to the bucket; then, takes the ball
Semaphores

```java
thread1

semaphore.wait()

If necessary, waits for a ball to be added to the bucket; then, takes the ball
```
Semaphores

`semaphore.wait()` (again)

If necessary, waits for a ball to be added to the bucket; then, takes the ball
Semaphores

semaphore.wait() (again)

If necessary, waits for a ball to be added to the bucket; then, takes the ball

thread1 (blocked)
Semaphores

semaphore.wait() (again)

thread1 (blocked)  thread2
Semaphores

Semaphore.wait() (again)

Semaphore.signal()

Adds a ball to the bucket, and wakes up any threads that were waiting for one to be added.

thread1 (blocked)

thread2
Semaphores

semaphore.wait() (again)

Adds a ball to the bucket, and wakes up any threads that were waiting for one to be added.

thread1 (blocked)

semaphore.signal()

thread2
Semaphores

semaphore.wait() (again)

thread1 is now unblocked!

thread1 is now unblocked!

Adds a ball to the bucket, and wakes up any threads that were waiting for one to be added.
Semaphores

semaphore.wait() (again)

Adds a ball to the bucket, and wakes up any threads that were waiting for one to be added.
Semaphores

semaphore.wait() (again)

thread1

thread2
Semaphores
Semaphore methods

- **signal():**
  - Adds a ball to the bucket
  - *Never blocks*

- **wait():**
  - If a ball is in the bucket, takes the ball and returns immediately
  - If no ball is in the bucket, waits until one is available, then takes the ball and returns

- There isn’t anything *actually* stored in the bucket. (Under the hood, semaphores are implemented with a simple counter indicating how many “balls” (or whatever) are in the bucket.) But they are very useful for synchronizing between threads.
Producer-consumer: transferring data between threads

- **Mutex**: Unlocked
- **Buffer**:
  ```
  SomeStruct {
    ...
  }
  ```
Producer-consumer: transferring data between threads

```
SomeStruct {
...
}
```

Mutex: Unlocked
Buffer:

```
semaphore.wait()
```

thread1
Producer-consumer: transferring data between threads

```java
semaphore.wait()
```

Thread 1

Mutex: Unlocked

Buffer:

```java
SomeStruct {
    ...
}
```
Producer-consumer: transferring data between threads

Semaphore: `wait()`

Mutex: Unlocked

Buffer: `SomeStruct { ... }`
Producer-consumer: transferring data between threads

```
mutex.lock()

Mutex: Unlocked
```

```
SomeStruct {
  …
}
```

Buffer:
Producer-consumer: transferring data between threads

mutex.lock()
Producer-consumer: transferring data between threads

SomeStruct {
    ...
}

Mutex: Locked
Buffer:
Producer-consumer: transferring data between threads

SomeStruct {
  ...
}

mutex.unlock()

Mutex: Locked
Buffer:
Producer-consumer: transferring data between threads

```cpp
SomeStruct {
    ...
}
```

```cpp
mutex.unlock() thread1
```

Mutex: Unlocked Buffer:
Producer-consumer: transferring data between threads

semaphore.wait() (again)

SomeStruct {
...
}

Mutex: Unlocked

Buffer:
Producer-consumer: transferring data between threads

```
SomeStruct {
    ...
}
```

Semaphore.wait() (again)

Thread1 (blocked)

Mutex: Unlocked  Buffer:
Producer-consumer: transferring data between threads

Semaphore.wait() (again)

SomeStruct {
  ...
}

Mutex: Unlocked
Buffer:

thread1 (blocked)

thread2

SomeStruct {
  ...
}
Producer-consumer: transferring data between threads

SomeStruct {
    ...
}

semaphore.wait() (again)

Mutex: Unlocked

Buffer:

Mutex: Unlocked

Buffer:

thread1 (blocked)

mutex.lock()
Producer-consumer: transferring data between threads

Semaphore.wait() (again)

SomeStruct {
  ...
 }

Mutex: Locked

Buffer:

thread1 (blocked)

thread2

SomeStruct {
  ...
 }

mutex.lock()
Producer-consumer: transferring data between threads

Semaphore.wait() (again)

SomeStruct {
  ...
}

Mutex: Locked

Buffer: SomeStruct {
  ...
}

thread1 (blocked)

thread2
Producer-consumer: transferring data between threads

```
SomeStruct {
    ...
}

Buffer:
```

Mutex: Locked

Semaphore.wait() (again)

mutex.unlock()

thread1 (blocked)

thread2

Producer-consumer: transferring data between threads
Producer-consumer: transferring data between threads

```cpp
SomeStruct {
    ...
}

Buffer:

Mutex: Unlocked

Semaphore.wait() (again)

thread1 (blocked)

mutex.unlock()

thread2

Producer-consumer: transferring data between threads
```
Producer-consumer: transferring data between threads

Semaphore.wait() (again)

Semaphore.signal()

SomeStruct {
  ...
}

Mutex: Unlocked

Buffer:
  SomeStruct {
    ...
  }

Thread1 (blocked)

Thread2
Producer-consumer: transferring data between threads

SomeStruct {
  ...
}

Mutex: Unlocked
Buffer: SomeStruct {
  ...
}

Semaphore.wait() (again)
Semaphore.signal()
Producer-consumer: transferring data between threads

`semaphore.wait()` (again)

```c
SomeStruct {
    ...
}
```

Mutex: Unlocked  Buffer: 

```c
SomeStruct {
    ...
}
```
Producer-consumer: transferring data between threads

SomeStruct {
...
}

Buffer:

Mutex: Unlocked

Semaphore.wait() (again)

thread1

thread2

SomeStruct {
...
}
Producer-consumer: transferring data between threads

```
mutex.lock()

SomeStruct {
  ...
}
```

Mutex: Unlocked  Buffer: SomeStruct {
  ...
}
Producer-consumer: transferring data between threads

mutex.lock()

SomeStruct {
  ...
}

Buffer:

Mutex: Locked

thread1

thread2

SomeStruct {
  ...
}


Producer-consumer: transferring data between threads

SomeStruct {
    ...
}

SomeStruct {
    ...
}

Mutex: Locked

Buffer:
Producer-consumer: transferring data between threads

SomeStruct {
...
}

SomeStruct {
...
}

Mutex: Locked

Buffer:

mutex.unlock()
Producer-consumer: transferring data between threads

mutex.unlock()

SomeStruct {
    ...
}

SomeStruct {
    ...
}

Mutex: Unlocked

Buffer:

thread1

thread2