Missed Signals

Missed Signals

A missed signal happens when a signal is sent by a thread before the other thread starts waiting on a condition. This is exemplified by the following code snippet. Missed signals are caused by using the wrong concurrency constructs. In the example below, a condition variable is used to coordinate between the **signaller** and the **waiter** thread. The condition is signaled at a time when no thread is waiting on it causing a missed signal.

In later sections, you'll learn that the way we are using the condition variable's `await` method is incorrect. The idiomatic way of using `await` is in a while loop with an associated boolean condition. For now, observe the possibility of losing signals between threads.

```
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.ReentrantLock;

class Demonstration {

    public static void main(String args[]) throws InterruptedException {
        MissedSignalExample.example();
    }
}

class MissedSignalExample {

    public static void example() throws InterruptedException {

        final ReentrantLock lock = new ReentrantLock();
        final Condition condition = lock.newCondition();

        Thread signaller = new Thread(new Runnable() {

            public void run() {
                lock.lock();
```

```
                condition.signal();
                System.out.println("Sent signal");
                lock.unlock();

            }
        });

        Thread waiter = new Thread(new Runnable() {

            public void run() {

                lock.lock();

                try {
                    condition.await();
                    System.out.println("Received signal");
                } catch (InterruptedException ie) {
                    // handle interruption
                }

                lock.unlock();

            }
        });

        signaller.start();
        signaller.join();

        waiter.start();
        waiter.join();

        System.out.println("Program Exiting.");
    }
}
```




Missed Signal Example

The above code when ran, will never print the statement `Program Exiting` and execution would time out. Apart from refactoring the code to match the idiomatic usage of condition variables in a while loop, the other possible fix is to use a **semaphore** for signalling between the two threads as shown below

```
import java.util.concurrent.Semaphore;

class Demonstration {

    public static void main(String args[]) throws InterruptedException {
        FixedMissedSignalExample.example();
```

```
    }
}


class FixedMissedSignalExample {

    public static void example() throws InterruptedException {

        final Semaphore semaphore = new Semaphore(1);

        Thread signaller = new Thread(new Runnable() {

            public void run() {
                semaphore.release();
                System.out.println("Sent signal");
            }
        });

        Thread waiter = new Thread(new Runnable() {

            public void run() {
                try {
                    semaphore.acquire();
                    System.out.println("Received signal");
                } catch (InterruptedException ie) {
                    // handle interruption
                }
            }
        });

        signaller.start();
        signaller.join();
        Thread.sleep(5000);
        waiter.start();
        waiter.join();

        System.out.println("Program Exiting.");
    }
}
```




Fixed Missed Signal