Printing Number Series (Zero, Even, Odd)

This problem is about repeatedly executing threads which print a specific type of number. Another variation of this problem; print even and odd numbers; utilizes two threads instead of three.

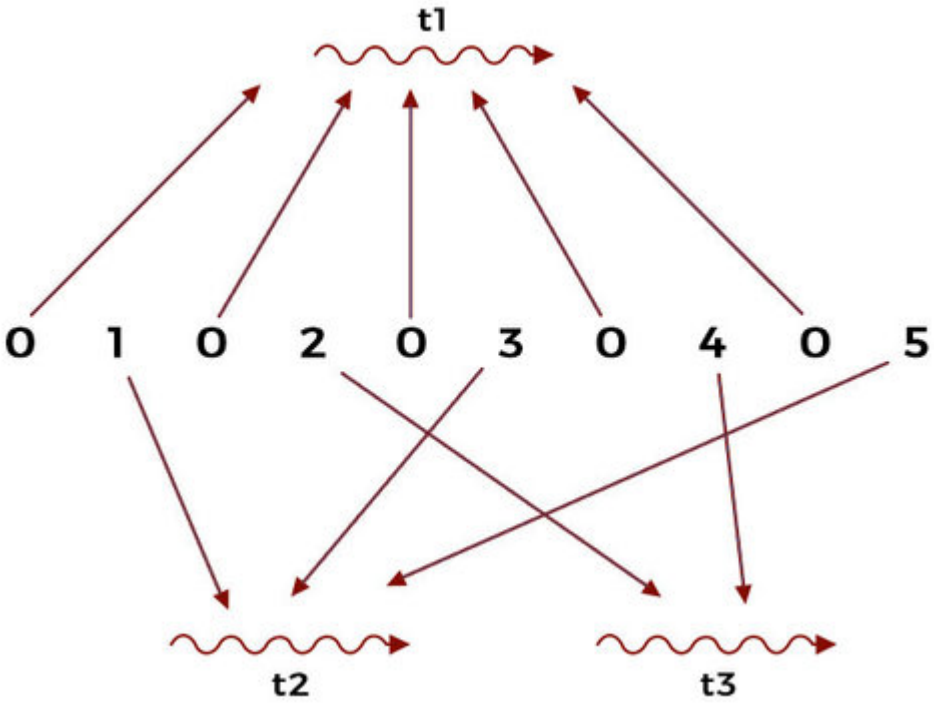
Problem

Suppose we are given a number *n* based on which a program creates the series *010203...0n*. There are three threads t1, t2 and t3 which print a specific type of number from the series. t1 only prints zeros, t2 prints odd numbers and t3 prints even numbers from the series. The code for the class is given as follows:

```
class PrintNumberSeries
    #constructor
    def initialize(n)
        @n = n
    end

    def PrintZero
    end

    def PrintOdd
    end

    def PrintEven
    end

end
```

You are required to write a program which takes a user input n and outputs the number series using three threads. The three threads work together to print zero, even and odd numbers. The threads should be synchronized so that the functions PrintZero(), PrintOdd() and PrintEven() are executed in an order.

The workflow of the program is shown below:



Workflow

Solution

This problem is solved by using Semaphores in the concurrent library of Ruby. Concurrent Ruby is the only Ruby library which provides a full set of thread safe variable types and data structures. The basic structure of the class is given below:

```
class PrintNumberSeries

    def initialize(n)
    end

    def PrintZero
    end

    def PrintOdd
    end

    def PrintEven
    end
```

```
end
```

The solution makes use of three semaphores; `zeroSem` for printing zeros, `oddSem` for printing odd numbers and `evenSem` for printing even numbers. Semaphores are used to restrict the number of threads than can access some (physical or logical) resource at the same time. The constructor of this class appears below:

```
def initialize(n)
        @n = n
        @zeroSem = Concurrent::Semaphore.new(1)
        @oddSem = Concurrent::Semaphore.new(0)
        @evenSem = Concurrent::Semaphore.new(0)
end
```

`n` is the user input that prints the series till *n*th number. The argument passed to semaphore's constructor is the number of 'permits' available. For `oddSem` and `evenSem`, all `acquire()` calls will be blocked initially as they are initialized with 0. For `zeroSem`, the first `acquire()` call will succeed as it is intialized with 1. The code of the first method `PrintZero()` is as follows:

```
def PrintZero
        i = 0
        while i < @n
                @zeroSem.acquire
                print "0"
                if (i % 2 == 0)
                        @oddSem.release
                else
                        @evenSem.release
                end
                i += 1
        end
end
```

`PrintZero()` begins with a loop iterating from 0 till `n` (exclusive). The semaphore `zeroSem` is acquired and '0' is printed. A very significant line in this method is **line 6** in the loop. The modulus operator (**%**) gives the remainder of a division by the value following it. In our case, the current value is divided by 2 to determine if `i` is even or odd. If `i` is odd, then it

means we just printed an odd number and the next number in the sequence will be an even number so, `evenSem` is released. In the same way if `i` is even then `oddSem` is released for printing the next odd number in the sequence. The second method `PrintOdd()` is shown below:

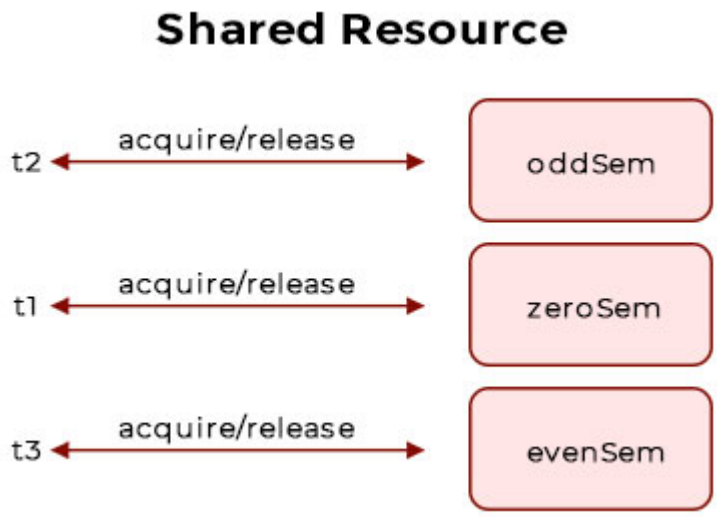
```
def PrintOdd
    i = 1
    while i <= @n
        @oddSem.acquire
        print "#{i}"
        @zeroSem.release
        i += 2
    end
end
```

The loop iterates from 1 till `n` (inclusive) and `i` is incremented by 2 after each iteration to ensure that only odd numbers are printed. `oddSem` is acquired when `PrintZero()` releases it after determining that it is the turn for an odd number to be printed. Since zero is required to be printed before every even/odd number, `zeroSem` is released after printing the odd number. The last method of the class `PrintEven()` is shown below:

```
def PrintEven
    i = 2
    while i <= @n
        @evenSem.acquire
        print "#{i}"
        @zeroSem.release
        i += 2
    end
end
```

`PrintEven()` operates in the same manner as `PrintOdd()` except that its loop begins from 2. `evenSem` is acquired if released by `PrintZero()` and an even number is printed. `zeroSem` is released for zero to be printed next. If `n` is reached, the loop breaks.

The working of this class can be seen as a lock-shift phenomenon where every method is given the control at its turn and blocked otherwise. `n` is manipulated by only one thread at a time.



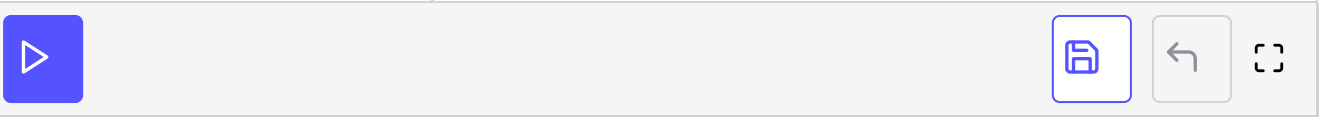
To test our solution, we will be reusing `CountingSemaphore` class that we implemented earlier. A total of 3 threads are created; `t1`, `t2` and `t3`. `t1` prints 0, `t2` prints odd numbers and `t3` prints even numbers. The threads are started in random order.

main.rb

CountingSemaphore.rb

```
require './CountingSemaphore'
class PrintNumberSeries

        def initialize(n)
                @n = n
                @zeroSem = CountingSemaphore.new(1)
                @oddSem = CountingSemaphore.new(1)
                @evenSem = CountingSemaphore.new(1)
                @oddSem.acquire
                @evenSem.acquire
        end

        def PrintZero
                i = 0
                while i < @n
                        @zeroSem.acquire
                        print "0"
                        if (i % 2 == 0)
                                @oddSem.release
                        else
                                @evenSem.release
                        end
                        i += 1
                end
        end

        def PrintOdd
                i = 1
                while i <= @n
```

```
                        @oddSem.acquire
                        print "#{i}"
                        @zeroSem.release

                        i += 2
                end
        end

        def PrintEven
                i = 2
                while i <= @n
                        @evenSem.acquire
                        print "#{i}"
                        @zeroSem.release
                        i += 2
                end
        end
end

zeo = PrintNumberSeries.new(5)

t1 = Thread.new do
                zeo.PrintZero
        end
t2 = Thread.new do
                zeo.PrintOdd
        end
t3 = Thread.new do
                zeo.PrintEven
end
[t2, t3, t1].each(&:join)
```



> We were told to use three threads in the problem statement but the solution can be achieved using two threads as well. Since zero is printed before every number, we do not need to dedicate a special thread for it. We can simply print a zero before printing every odd or even number.