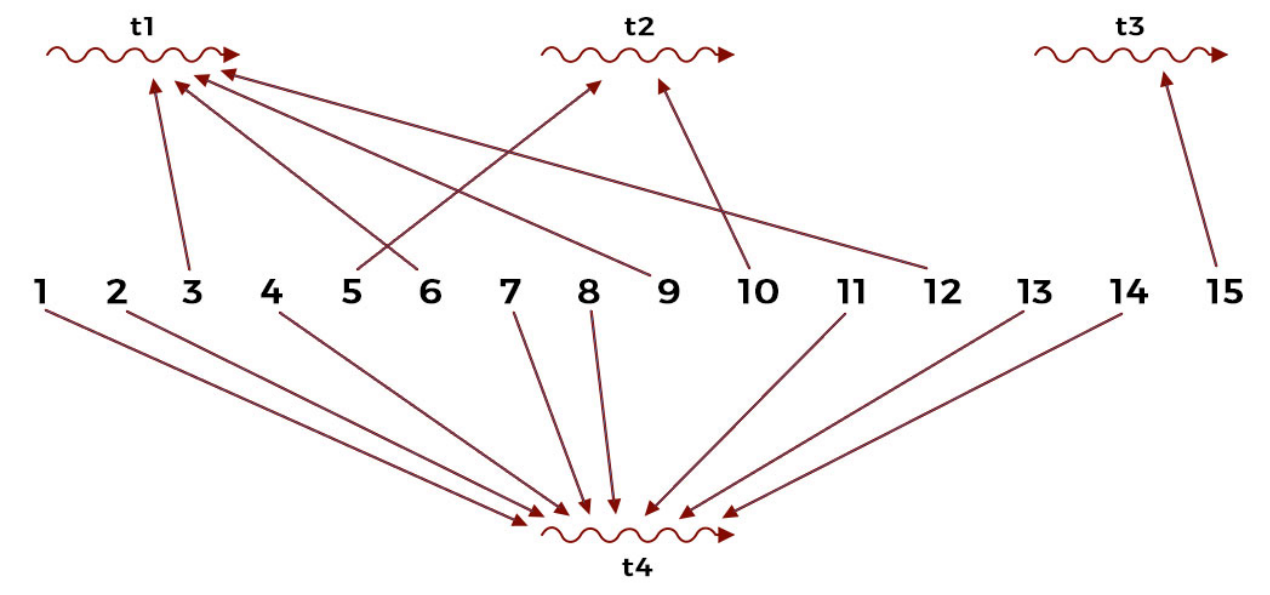
Fizz Buzz Problem

This problem explores a multi-threaded solution to the very common Fizz Buzz programming task

Problem

FizzBuzz is a common interview problem in which a program prints a number series from 1 to n such that for every number that is a multiple of 3 it prints "fizz", for every number that is a multiple of 5 it prints "buzz" and for every number that is a multiple of both 3 and 5 it prints "fizzbuzz". We will be creating a multi-threaded solution for this problem.

Suppose we have four threads t1, t2, t3 and t4. Thread t1 checks if the number is divisible by 3 and prints **fizz**. Thread t2 checks if the number is divisible by 5 and prints **buzz**. Thread t3 checks if the number is divisible by both 3 and 5 and prints **fizzbuzz**. Thread t4 prints numbers that are not divisible by 3 or 5. The workflow of the program is shown below:



The code for the class is as follows:

```
class MultithreadedFizzBuzz {
```

```
    def initialize(n)
        @n = n

    end

    def fizz
        puts "fizz"
    end

    def buzz
        puts "buzz"
    end

    def fizzbuzz
        puts "fizzbuzz"
    end

    def Number()
        puts "#{@num}"
    end
}
```

For an input integer n, the program should output a string containing the words fizz, buzz and fizzbuzz representing certain numbers. For example, for n = 15, the output should be: 1, 2, fizz, 4, buzz, fizz, 7, 8, fizz, buzz, 11, fizz, 13, 14, fizzbuzz.

Solution

We will solve this problem using the basic Ruby synchronization utilities; Mutex and Condition Variable. The basic structure of the class is given below.

```
class MultithreadedFizzBuzz

    def initialize(n)
        @n = n
        @num = 1
        @mutex = Mutex.new
        @cv = ConditionVariable.new
    end
```

```
    def fizzbuzz
    end

    def fizz
    end

    def buzz
    end

    def number
    end

end
```

The `MultithreadedFizzBuzz` class contains 4 private members: `n`, `num`, `mutex` and `cv`.

`n` is the last number of the series to be printed whereas `num` represents the current number to be printed. It is initialized with 1. `mutex` is used for synchronization amongst the threads and `cv` is a condition variable used for envoking `wait()` and `broadcast()` for the threads. The constructor method shown below initializes `n` with the input taken from user.

```
def initialize(n)
    @n = n
    @num = 1
    @mutex = Mutex.new
    @cv = ConditionVariable.new
end
```

The second method in the class, `fizz` prints "fizz" only if the current number is divisible by 3. The first loop checks if `num` (current number) is smaller than or equal to `n` (user input). Then `num` is checked for its divisibility by 3. We check if `num` is divisible by 3 and not by 5 because some multiples of 3 are also multiples of 5. If the condition is met, then "fizz" is printed and `num` is incremented. The waiting threads are notified via `broadcast`. If the condition is not met, the thread goes into `wait`.

```
def fizz
    @mutex.synchronize do
        while(@num <= @n)
```

```
            if (@num % 3 == 0 && @num % 5 != 0)
                puts "fizz"

                @num += 1
                @cv.broadcast
            else
                @cv.wait(@mutex)
            end
        end
    end
end
```

The next method `buzz` works in the same manner as `fizz`. The only difference here is the check to see if `num` is divisible by 5 and not by 3. The reasoning is the same: some multiples of 5 are also multiples of 3 and those numbers should not be printed by this function. If the condition is met, then "buzz" is printed otherwise the thread will `wait`.

```
def buzz
    @mutex.synchronize do
        while(@num <= @n)
            if (@num % 5 == 0 && @num % 3 != 0)
                puts "buzz"
                @num += 1
                @cv.broadcast
            else
                @cv.wait(@mutex)
            end
        end
    end
end
```

The next method `fizzbuzz` prints "fizzbuzz" if the current number in the series is divisible by both 3 and 5. A multiple of 15 is divisible by 3 and 5 both so `num` is checked for its divisibility by 15. After printing "fizzbuzz", `num` is incremented and waiting threads are notified via `broadcast`. If `num` is not divisible by 15 then the calling thread will `wait`.

```
def fizzbuzz
    @mutex.synchronize do
        while(@num <= @n)
            if (@num % 15 == 0)
                puts "fizzbuzz"
                @num += 1
```

```
                @cv.broadcast
            else
                @cv.wait(@mutex)
            end
        end
    end
end
```

The last method `number` checks if `num` is neither divisible by 3 nor by 5, then print the `num`.

```
def number
    @mutex.synchronize do
        while(@num <= @n)
            if (@num % 5 != 0 && @num % 3 != 0)
                puts "#{@num}"
                @num += 1
                @cv.broadcast
            else
                @cv.wait(@mutex)
            end
        end
    end
end
```

The complete code for `MultithreadedFizzBuzz` is as follows:

```
class MultithreadedFizzBuzz

    def initialize(n)
        @n = n
        @num = 1
        @mutex = Mutex.new
        @cv = ConditionVariable.new
    end

    def fizz
        @mutex.synchronize do
            while(@num <= @n)
                if (@num % 3 == 0 && @num % 5 != 0)
                    puts "fizz"
                    @num += 1
                    @cv.broadcast
                else
                    @cv.wait(@mutex)
```

```
            end
        end
    end
end

def buzz
    @mutex.synchronize do
        while(@num <= @n)
            if (@num % 5 == 0 && @num % 3 != 0)
                puts "buzz"
                @num += 1
                @cv.broadcast
            else
                @cv.wait(@mutex)
            end
        end
    end
end

def fizzbuzz
    @mutex.synchronize do
        while(@num <= @n)
            if (@num % 15 == 0)
                puts "fizzbuzz"
                @num += 1
                @cv.broadcast
            else
                @cv.wait(@mutex)
            end
        end
    end
end

def number
    @mutex.synchronize do
        while(@num <= @n)
            if (@num % 3 != 0 && @num % 5 != 0)
                puts "#{@num}"
                @num += 1
                @cv.broadcast
            else
                @cv.wait(@mutex)
            end
        end
    end
end
```

```
end
```

To test our solution, we will be making 4 threads: **t1**,**t2**, **t3** and **t4**. Three threads will check for divisibility by 3, 5 and 15 and print **fizz**, **buzz**, and **fizzbuzz** accordingly.. Thread **t4** prints numbers that are not divisible by 3 or 5.

```
class MultithreadedFizzBuzz

        def initialize(n)
                @n = n
                @num = 1
                @mutex = Mutex.new
                @cv = ConditionVariable.new
        end

        def fizz
                @mutex.synchronize do
                        while(@num <= @n)
                                if (@num % 3 == 0 && @num % 5 != 0)
                                        puts "fizz"
                                        @num += 1
                                        @cv.broadcast
                                else
                                        @cv.wait(@mutex)
                                end
                        end
                end
        end

        def buzz
                @mutex.synchronize do
                        while(@num <= @n)
                                if (@num % 5 == 0 && @num % 3 != 0)
                                        puts "buzz"
                                        @num += 1
                                        @cv.broadcast
                                else
                                        @cv.wait(@mutex)
                                end
                        end
                end
        end

        def fizzbuzz
                @mutex.synchronize do
                        while(@num <= @n)
                                if (@num % 15 == 0)
                                        puts "fizzbuzz"
                                        @num += 1
                                        @cv.broadcast
                                else
                                        @cv.wait(@mutex)
                                end
                        end
```

```
                end
        end

        def number
                @mutex.synchronize do
                        while(@num <= @n)
                                if (@num % 3 != 0 && @num % 5 != 0)
                                        puts "#{@num}"
                                        @num += 1
                                        @cv.broadcast
                                else
                                        @cv.wait(@mutex)
                                end
                        end
                end
        end
end

fb = MultithreadedFizzBuzz.new(15)


t1 = Thread.new(fb) do
        fb.fizz
    end
t2 = Thread.new(fb) do
        fb.buzz
    end
t3 = Thread.new(fb) do
        fb.fizzbuzz
    end
t4 = Thread.new(fb) do
        fb.number
    end

threads = []

threads << t3
threads << t1
threads << t2
threads << t4

threads.each(&:join)
```