… continued

This lesson continues the discussion on working with Fibers.

Passing and Returning from Fibers

We can pass and return values from fibers. Consider the snippet below:

```
fib = Fiber.new do |firstMsg|
  start = 0
  puts firstMsg

  while true do
    newMsg = Fiber.yield start
    puts newMsg
    start += 1
  end

end

10.times do |i|
  puts fib.resume("hello #{i}")
  sleep(1)
end
```

The interesting line in the above snippet is:

```
    newMsg = Fiber.yield start
```

The above statement can be conceptually broken down as the ordered sequence of the following operations:

1. Returns the `start` variable's value to the invoker of the fiber.
2. Relinquish control to the invoker of the fi

2. Relinquishes control to the invoker of the fiber.

3. Assigns the argument passed in by the invoker in the next `resume()` call to the variable `newMsg`.

Generating Infinite Lists/Sequences

One trivial application of fibers is to generate infinite lists or sequences. Consider the below snippet, that returns odd numbers on every `resume()`

```
fib = Fiber.new do
  odd = 1

  while true
    Fiber.yield odd
    odd += 2
  end

end

# print the first odd numbers.
10.times do
  puts fib.resume()
end
```

Transfer

A fiber can also transfer control to another fiber by invoking the `transfer()` method on the desired fiber object. For instance:

```
require 'fiber'

fib1 = fib2 = nil

fib2 = Fiber.new do |arg|
  while true
    puts "Control transferred to fiber2"
    Fiber.yield arg
  end
end

fib1 = Fiber.new do
```

```
  while true
    puts "Control transferred to fiber1"
    fib2.transfer(10)

  end
end

puts "Control in main thread"
puts "Received #{fib1.resume()} in main thread"
puts "Control back in main thread"
```

Running the above snippet you can see that we resume `fib1` in the main thread. The `fib1` fiber invokes the `transfer()` method on the `fib2` object and the control is passed to `fib2`. The `transfer()` method acts similar to the `resume()` method, in that, we can pass arguments to the target fiber. The `transfer()` either resumes the fiber for the first time or from the point it last suspended control. The main thread receives an integer 10 which is yielded from `fib2`. Note that invoking `Fiber.yield` returns control back to the main thread and not to `fib1`. What if we want to resume `fib1` a second time? Let's see below:

```
require 'fiber'

fib1 = fib2 = nil

fib2 = Fiber.new do |arg|
  while true
    puts "Control transferred to fiber2"
    Fiber.yield arg
  end
end

fib1 = Fiber.new do

  while true
    puts "Control transferred to fiber1"
    fib2.transfer(10)
  end
end

puts "Control in main thread"
puts "Received #{fib1.resume()}"
puts "Control back in main thread"

# Attempt to resume fib1 a second time
fib1.resume()
```

The widget throws a "double resume" error. The official documentation states that we can't resume a fiber **A** that transferred control to another fiber **B** before the control is transferred back to fiber **A**. However, if we try to do that as follows, we still get an error.

```
require 'fiber'

fib1 = fib2 = nil

fib2 = Fiber.new do
  while true
    puts "Control transferred to fiber2"
    # transfer controll back to fib1
    fib1.transfer()
  end
end

fib1 = Fiber.new do

  while true
    puts "Control transferred to fiber1"
    fib2.transfer()
    Fiber.yield
  end
end

puts "Control in main thread"
fib1.resume()
puts "Control back in main thread"

# Attempt to resume fib1 a second time
fib1.resume()
```

This functionality has been broken since MRI 2.0 and is a known bug being tracked here.