VIT®

CHENNAI

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

SWE2034 – Ruby Programming

Guided By – **Dr Yogesh C**

Slot – **L5+L6**

NAME: **VISHWANTH P**

REGISTER.NO: **21MIS1117**

Lab Assessment - 5

1) **Write a separate program using the following functions:**

Fiber – yield and resume

Fiber – transfer

Fiber – raise

Code

```ruby
fiber1 = Fiber.new do
  puts "Fiber1 is running"
  Fiber.yield
  puts "Fiber1 is resumed"
end

fiber1.resume
puts "back to Main"
fiber1.resume

fiber2 = Fiber.new do
  puts "Fiber2 is running"
end

fiber3 = Fiber.new do
  puts "Fiber3 is running"
  fiber2.transfer
end

fiber3.resume

fiber4 = Fiber.new do
  begin
    Fiber.yield
  rescue => e
    puts "Caught an exception: #{e.message}"
  end
end

fiber4.resume
fiber4.raise "An error occurred in Fiber4"
```

Output:

```
PS D:\7th Sem\F1 - Ruby\Lab\LAB_5> ruby Q1.rb
Fiber1 is running
back to Main
Fiber1 is resumed
Fiber3 is running
Fiber2 is running
Caught an exception: An error occurred in Fiber4
PS D:\7th Sem\F1 - Ruby\Lab\LAB_5>
```

2) Create 10 threads, each of which sleep for a random amount of time and then prints a message.

Code

```ruby
require 'thread'
threads = []
10.times do |i|
    threads << Thread.new do
        sleep_time = rand(1..5)
        sleep(sleep_time)
        puts "Thread #{i+1} woke up later #{sleep_time} seconds."
    end
end
threads.each(&:join)
```

Output

```
PS D:\7th Sem\F1 - Ruby\Lab\LAB_5> ruby Q2.rb
Thread 3 woke up later 1 seconds.
Thread 6 woke up later 1 seconds.
Thread 10 woke up later 2 seconds.
Thread 9 woke up later 2 seconds.
Thread 1 woke up later 3 seconds.
Thread 8 woke up later 3 seconds.
Thread 7 woke up later 3 seconds.
Thread 2 woke up later 4 seconds.
Thread 5 woke up later 4 seconds.
Thread 4 woke up later 5 seconds.
PS D:\7th Sem\F1 - Ruby\Lab\LAB_5>
```

3) Create a local variable for a main thread, additional threads and fiber and prints the value of it.

Code

```ruby
Main_var = "Main thread variable"
Fiber_var = "Fiber variable"
Thread_var = "Thread variable"
fiber = Fiber.new do
    puts " inside fiber : #{Fiber_var}"
end
thread = Thread.new do
    puts " inside thread :  #{Thread_var}"
end
puts " in main thread : #{Main_var}"
fiber.resume
thread.join
```

Output

```
PS D:\7th Sem\F1 - Ruby\Lab\LAB_5> ruby Q3.rb
 in main thread : Main thread variable
 inside fiber : Fiber variable
 inside thread :  Thread variable
```

4) Local variable values in Nested Thread within a Fiber.

Code

```ruby
fiber = Fiber.new do
    local_fiber_var = "Fiber variable"
    Thread.new do
        thread_var = "Thread variable"
        puts thread_var
        puts local_fiber_var
    end.join
end

fiber.resume
```

Output

```
PS D:\7th Sem\F1 - Ruby\Lab\LAB_5> ruby Q4.rb
Thread variable
Fiber variable
```

5) Local variable values in Nested Fiber within a Thread.

Code

```ruby
Thread.new do
    thread_var = "Thread variable"
    fiber=Fiber.new do
        local_fiber_var = "fiber variable"
        puts local_fiber_var
        puts thread_var
    end
    fiber.resume
end.join
```

Output

```
PS D:\7th Sem\F1 - Ruby\Lab\LAB_5> ruby Q5.rb
fiber variable
Thread variable
```

6) Multi Thread sharing same variable address space.

Code

```ruby
shared_var = 0
mutex = Mutex.new
5.times do |i|
  Thread.new do
    mutex.synchronize do
      shared_var += 1
      puts "Thread #{i + 1} increased shared_var to #{shared_var}"
    end
  end
end
Thread.list.each { |t| t.join unless t == Thread.main }
```

Output

```
PS D:\7th Sem\F1 - Ruby\Lab\LAB_5> ruby Q6.rb
Thread 1 increased shared_var to 1
Thread 2 increased shared_var to 2
Thread 3 increased shared_var to 3
Thread 4 increased shared_var to 4
Thread 5 increased shared_var to 5
```

7) **Write a separate program using the following functions:**

a. **Thread – Stop and Run**

Code

```ruby
t = Thread.new do
    puts "Thread running, then stopping"
    Thread.stop
    puts "Thread resumed"
  end
  sleep 1
  puts "Main thread waking up the stopped thread"
  t.run
  t.join
```

Output

```
PS D:\7th Sem\F1 - Ruby\Lab\LAB_5\Q7> ruby a.rb
Thread running, then stopping
Main thread waking up the stopped thread
Thread resumed
```

b. **Thread – Wakeup**

Code

```ruby
t = Thread.new do
    puts "Thread sleeping"
    Thread.stop
    puts "Thread woke up"
  end

  sleep 1
  puts "Main thread waking up the stopped thread"
  t.wakeup
  t.join
```

Output

```
PS D:\7th Sem\F1 - Ruby\Lab\LAB_5\Q7> ruby b.rb
Thread sleeping
Main thread waking up the stopped thread
Thread woke up
```

c. Thread – Value

Code

```ruby
t = Thread.new do
  sleep 2
  70
end
puts "Thread returned value: #{t.value}"
```

Output

```
PS D:\7th Sem\F1 - Ruby\Lab\LAB_5\Q7> ruby c.rb
Thread returned value: 70
```

d. Thread – Pass

Code

```ruby
t1 = Thread.new { puts "Thread 1 executing" }
t2 = Thread.new { puts "Thread 2 executing" }

Thread.pass
t1.join
t2.join
```

Output

```
PS D:\7th Sem\F1 - Ruby\Lab\LAB_5\Q7> ruby d.rb
Thread 1 executing
Thread 2 executing
```

e. Thread –Priority

Code

```ruby
t1 = Thread.new { sleep 1; puts "Thread with lower priority" }
t2 = Thread.new { puts "Thread with higher priority" }

t1.priority = -1
t2.priority = 1

t1.join
t2.join
```

Output

```
PS D:\7th Sem\F1 - Ruby\Lab\LAB_5\Q7> ruby e.rb
Thread with higher priority
Thread with lower priority
```

f. Thread – Mutex

Code

```ruby
mutex = Mutex.new
counter = 0

5.times do |i|
  Thread.new do
    mutex.synchronize do
      counter += 1
      puts "Thread #{i + 1} increased counter to #{counter}"
    end
  end
end

Thread.list.each { |t| t.join unless t == Thread.main }
```

Output

```
PS D:\7th Sem\F1 - Ruby\Lab\LAB_5\Q7> ruby f.rb
Thread 1 increased counter to 1
Thread 2 increased counter to 2
Thread 3 increased counter to 3
Thread 4 increased counter to 4
Thread 5 increased counter to 5
```

g. Thread – Fork

Code

```ruby
t = Thread.fork do
    puts "Thread created using fork"
end
t.join
```

Output

```
PS D:\7th Sem\F1 - Ruby\Lab\LAB_5\Q7> ruby g.rb
Thread created using fork
```