

Ruby Programming Lab 5

Suryakumar P 21MIS1146

1. Write separate program using:
 - a. Yield and resume

Code:

```
counter = Fiber.new do
  count = 0
  loop do
    count += 1
    Fiber.yield(count)
  end
end

# Example 1: Basic resume and yield
puts "Example 1: Basic counting"
3.times do
  puts "Count: #{counter.resume}"
end
```

Output:

```
PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week6> ruby yield.rb
Basic counting
Count: 1
Count: 2
Count: 3
PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week6> █
```

- b. Transfer

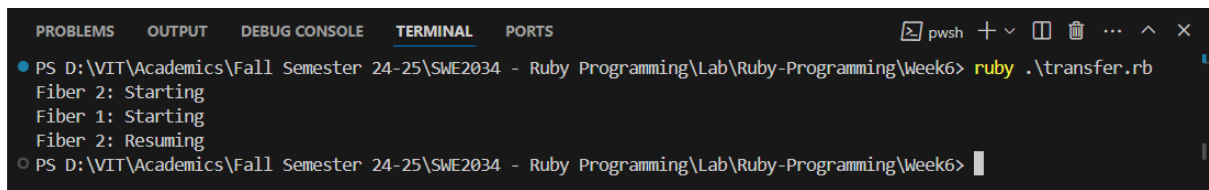
Code:

```
fiber1 = Fiber.new do
  puts "Fiber 1: Starting"
  Fiber.yield
  puts "Fiber 1: Resuming"
end

fiber2 = Fiber.new do
  puts "Fiber 2: Starting"
  fiber1.resume
  puts "Fiber 2: Resuming"
end

fiber2.resume
```

Output:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week6> ruby .\transfer.rb
Fiber 2: Starting
Fiber 1: Starting
Fiber 2: Resuming
PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week6>
```

c. Raise

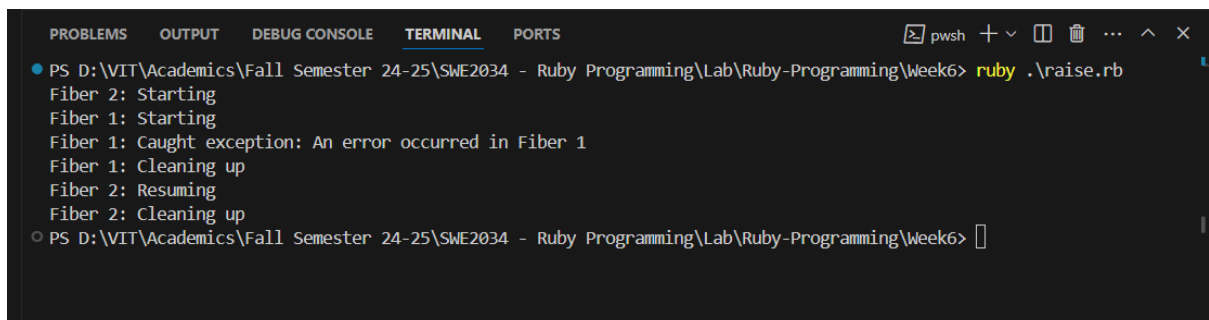
Code:

```
fiber1 = Fiber.new do
  begin
    puts "Fiber 1: Starting"
    raise "An error occurred in Fiber 1"
  rescue => e
    puts "Fiber 1: Caught exception: #{e.message}"
  ensure
    puts "Fiber 1: Cleaning up"
  end
end

fiber2 = Fiber.new do
  begin
    puts "Fiber 2: Starting"
    fiber1.resume
    puts "Fiber 2: Resuming"
  rescue => e
    puts "Fiber 2: Caught exception: #{e.message}"
  ensure
    puts "Fiber 2: Cleaning up"
  end
end

fiber2.resume
```

Output:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week6> ruby .\raise.rb
Fiber 2: Starting
Fiber 1: Starting
Fiber 1: Caught exception: An error occurred in Fiber 1
Fiber 1: Cleaning up
Fiber 2: Resuming
Fiber 2: Cleaning up
PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week6>
```

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

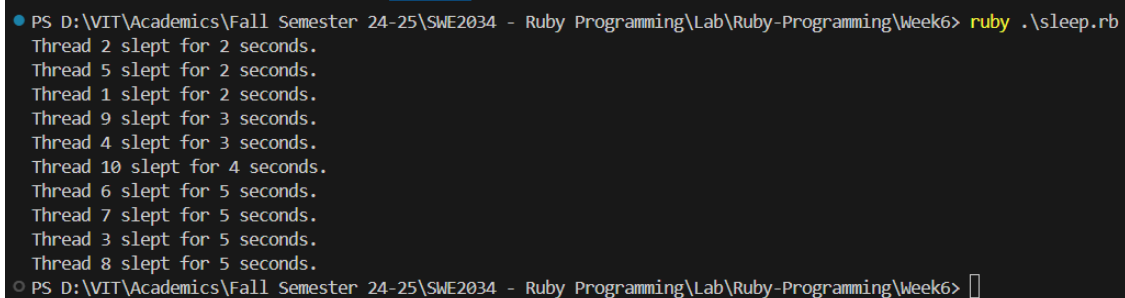
Code:

```
threads = []

10.times do |i|
  threads << Thread.new do
    sleep_duration = rand(1..5)
    sleep(sleep_duration)
    puts "Thread #{i + 1} slept for #{sleep_duration} seconds."
  end
end

threads.each(&:join)
```

Output:



```
PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week6> ruby .\sleep.rb
Thread 2 slept for 2 seconds.
Thread 5 slept for 2 seconds.
Thread 1 slept for 2 seconds.
Thread 9 slept for 3 seconds.
Thread 4 slept for 3 seconds.
Thread 10 slept for 4 seconds.
Thread 6 slept for 5 seconds.
Thread 7 slept for 5 seconds.
Thread 3 slept for 5 seconds.
Thread 8 slept for 5 seconds.
PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week6>
```

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

Code:

```
main_thread_variable = "Main Thread Variable"

fiber = Fiber.new do
  fiber_variable = "Fiber Variable"
  puts "Inside Fiber: #{fiber_variable}"
  puts "Inside Fiber accessing main thread variable: #{main_thread_variable}"
end

threads = 3.times.map do |i|
  Thread.new do
    thread_variable = "Thread #{i + 1} Variable"
    puts "Inside Thread #{i + 1}: #{thread_variable}"
    puts "Inside Thread #{i + 1} accessing main thread variable:
#{main_thread_variable}"
  end
end
```

```

fiber.resume

threads.each(&:join)

puts "In Main Thread: #{main_thread_variable}"

```

Output:

```

● PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week6> ruby Q3.rb
Inside Fiber: Fiber Variable
Inside Fiber accessing main thread variable: Main Thread Variable
Inside Thread 1: Thread 1 Variable
Inside Thread 1 accessing main thread variable: Main Thread Variable
Inside Thread 2: Thread 2 Variable
Inside Thread 2 accessing main thread variable: Main Thread Variable
Inside Thread 3: Thread 3 Variable
Inside Thread 3 accessing main thread variable: Main Thread Variable
In Main Thread: Main Thread Variable
○ PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week6>

```

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

Code:

```

# Create a fiber
fiber = Fiber.new do
  fiber_variable = "Fiber Variable"
  puts "Inside Fiber: #{fiber_variable}"

  # Create a thread within the fiber
  thread = Thread.new do
    thread_variable = "Thread Variable"
    puts "Inside Thread within Fiber: #{thread_variable}"
    puts "Inside Thread within Fiber accessing fiber variable:
#{fiber_variable}"
  end

  # Wait for the thread to complete
  thread.join
end

# Resume the fiber
fiber.resume

```

Output:

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week6> ruby Q4.rb
Inside Fiber: Fiber Variable
Inside Thread within Fiber: Thread Variable
Inside Thread within Fiber accessing fiber variable: Fiber Variable
○ PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week6>

```

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

Code:

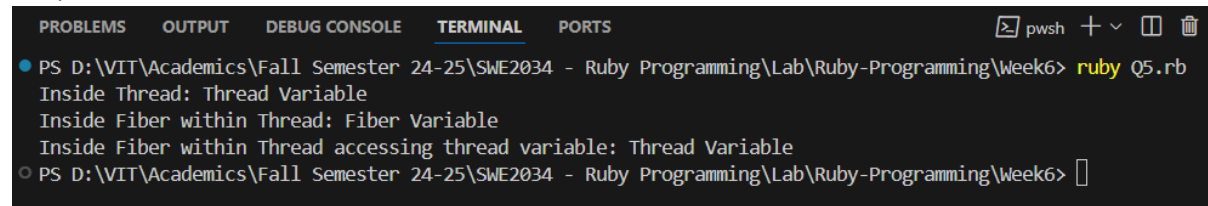
```
# Create a thread
thread = Thread.new do
  thread_variable = "Thread Variable"
  puts "Inside Thread: #{thread_variable}"

  # Create a fiber within the thread
  fiber = Fiber.new do
    fiber_variable = "Fiber Variable"
    puts "Inside Fiber within Thread: #{fiber_variable}"
    puts "Inside Fiber within Thread accessing thread variable:
#{thread_variable}"
  end

  # Resume the fiber
  fiber.resume
end

# Wait for the thread to complete
thread.join
```

Output:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week6> ruby Q5.rb
Inside Thread: Thread Variable
Inside Fiber within Thread: Fiber Variable
Inside Fiber within Thread accessing thread variable: Thread Variable
PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week6>
```

6. Multi Thread sharing same variable address space.

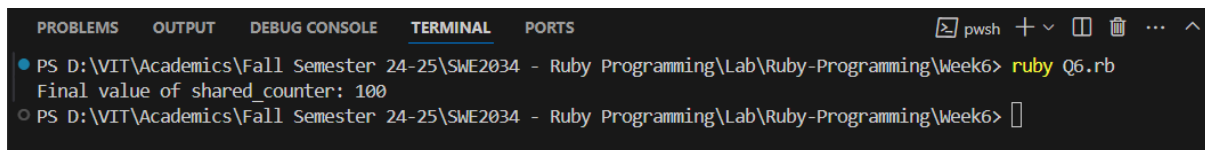
Code:

```
shared_counter = 0
mutex = Mutex.new

threads = 10.times.map do |i|
  Thread.new do
    10.times do
      mutex.synchronize do
        shared_counter += 1
      end
    end
  end
end

threads.each(&:join)
puts "Final value of shared_counter: #{shared_counter}"
```

Output:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week6> ruby Q6.rb
Final value of shared_counter: 100
PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week6>
```

7. Write a separate program using the following functions:

a. Thread – Stop and Run

Code:

```
running = true

thread = Thread.new do
  while running
    puts "Thread is running..."
    sleep(1)
  end
  puts "Thread has stopped."
end

sleep(3)
running = false

thread.join
puts "Main program has finished."
```

Output:

```
● PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week6> ruby Q7a.rb
Thread is running...
Thread is running...
Thread is running...
Thread has stopped.
Main program has finished.
○ PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week6> █
```

b. Thread – wake up

Code:

```
thread = Thread.new do
  puts "Thread is going to sleep..."
  sleep
  puts "Thread has been woken up!"
end

sleep(2)
puts "Main thread is waking up the sleeping thread..."

thread.wakeup

thread.join
```

Output:

```
● PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week6> ruby Q7b.rb
Thread is going to sleep...
Main thread is waking up the sleeping thread...
Thread has been woken up!
○ PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week6> █
```

c. Thread Value

Code:

```
thread = Thread.new do
  sum = 0
  1.upto(10) do |i|
    sum += i
  end
  sum
end

result = thread.value

puts "The sum of numbers from 1 to 10 is: #{result}"
```

Output:

```
● PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week6> ruby Q7c.rb
The sum of numbers from 1 to 10 is: 55
○ PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week6> █
```

d. Thread – pass

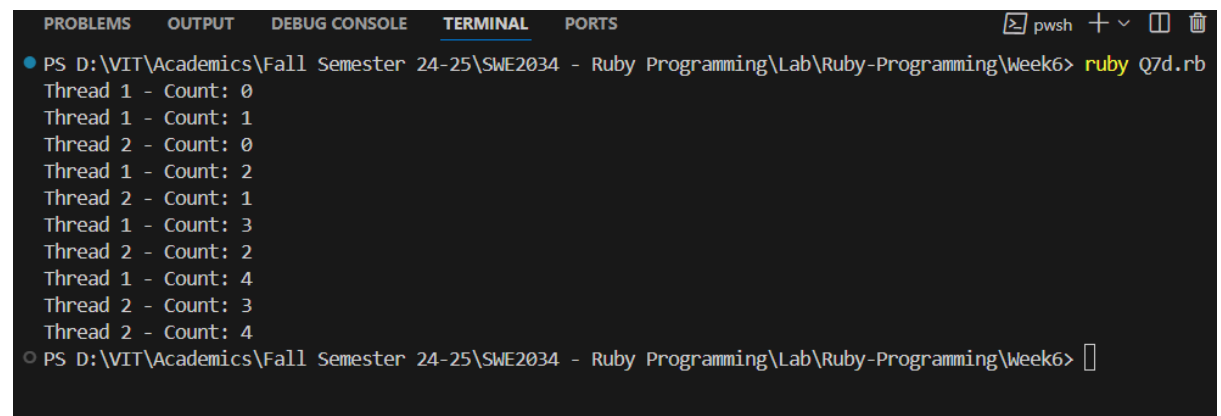
Code:

```
t1 = Thread.new do
  5.times do |i|
    puts "Thread 1 - Count: #{i}"
    Thread.pass
  end
end

t2 = Thread.new do
  5.times do |i|
    puts "Thread 2 - Count: #{i}"
    Thread.pass
  end
end

[t1, t2].each(&:join)
```

Output:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week6> ruby Q7d.rb
Thread 1 - Count: 0
Thread 1 - Count: 1
Thread 2 - Count: 0
Thread 1 - Count: 2
Thread 2 - Count: 1
Thread 1 - Count: 3
Thread 2 - Count: 2
Thread 1 - Count: 4
Thread 2 - Count: 3
Thread 2 - Count: 4
PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week6>
```


e. Thread – Priority

Code:

```
# Create a high-priority thread
high_priority_thread = Thread.new do
  5.times do
    puts "High priority thread is running"
    sleep(0.1)
  end
end

# Create a low-priority thread
low_priority_thread = Thread.new do
  5.times do
    puts "Low priority thread is running"
    sleep(0.1)
  end
end

# Set thread priorities
high_priority_thread.priority = 5
low_priority_thread.priority = 1

# Wait for both threads to complete
high_priority_thread.join
low_priority_thread.join
```

Output:

```
PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week6> ruby Q7e.rb
High priority thread is running
Low priority thread is running
High priority thread is running
Low priority thread is running
Low priority thread is running
High priority thread is running
High priority thread is running
Low priority thread is running
High priority thread is running
Low priority thread is running
PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week6> |
```

f. Thread – Mutex

Code:

```
shared_counter = 0

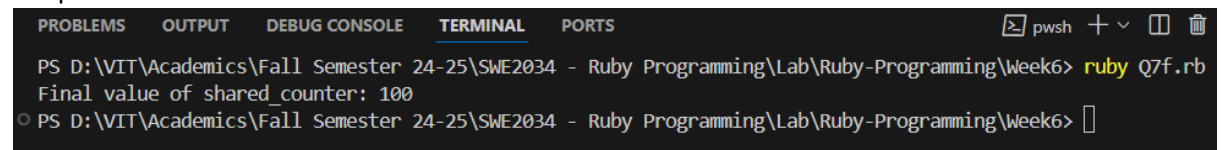
mutex = Mutex.new

threads = 10.times.map do |i|
  Thread.new do
    10.times do
      mutex.synchronize do
        shared_counter += 1
      end
    end
  end
end

threads.each(&:join)

puts "Final value of shared_counter: #{shared_counter}"
```

Output:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week6> ruby Q7f.rb
Final value of shared_counter: 100
PS D:\VIT\Academics\Fall Semester 24-25\SWE2034 - Ruby Programming\Lab\Ruby-Programming\Week6>
```

g. Thread – fork

Code:

```
pid = fork do
  5.times do |i|
    puts "Child process - Count: #{i}"
    sleep(0.5)
  end
end

if pid
  5.times do |i|
    puts "Parent process - Count: #{i}"
    sleep(0.5)
  end
  Process.wait(pid)
end
```

Output:

Fork works only on Unix based System

Parent process - Count: 0

Child process - Count: 0

Parent process - Count: 1

Child process - Count: 1

Parent process - Count: 2

Child process - Count: 2

Parent process - Count: 3

Child process - Count: 3

Parent process - Count: 4

Child process - Count: 4