RUBY LAB ASSESSMENT-5

21MIS1021 VIMAL KUMAR S

1. Write a seperate program using the following functions:
   Fiber – yield and resume
   Fiber – transfer
   Fiber – raise

CODE:

fiber

File Edit View

```
fiber1 = Fiber.new do
puts "Fiber 1 started"
value = Fiber.yeild("Hello from fiber 1")
puts "Fiber 1 received: #[value]"
raise "Fiber 1raised an exception"
puts "Fiber 1 resumed"
end

fiber2 = Fiber.new do
puts "Fiber 2 started"
value = fiber1.resume
puts "Fiber 2 received: #[value]"
fiber1.transfer("Hello from fiber 2")
puts "Fiber 2 resumed"
rescue => e
puts "Fiber 2 caught exception: #[e.message}"
end

puts fiber2.resume

puts "Program Finished"
```

OUTPUT:

```
C:\Users\student.PROGRAMMING404\Desktop\21MIS1021 VIMAL KUMAR S>ruby fiber.rb
Fiber 2 started
Fiber 1 started
Fiber 2 caught exception: #[e.message}

Program Finished

C:\Users\student.PROGRAMMING404\Desktop\21MIS1021 VIMAL KUMAR S>
```

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

CODE:

threads

File Edit View

```
threads = []

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

threads.each(&:join)
puts "All threads have completed."
```

OUTPUT:

```
C:\Users\student.PROGRAMMING404\Desktop\21MIS1021 VIMAL KUMAR S>ruby threads.rb
Thread #[index + 1} slept for 1 seconds.
Thread #[index + 1} slept for 1 seconds.
Thread #[index + 1} slept for 2 seconds.
Thread #[index + 1} slept for 3 seconds.
Thread #[index + 1} slept for 3 seconds.
Thread #[index + 1} slept for 4 seconds.
Thread #[index + 1} slept for 4 seconds.
Thread #[index + 1} slept for 4 seconds.
Thread #[index + 1} slept for 5 seconds.
Thread #[index + 1} slept for 5 seconds.
All threads have completed.

C:\Users\student.PROGRAMMING404\Desktop\21MIS1021 VIMAL KUMAR S>
```

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

File Edit View

```
main_thread = Thread.current
additional_threads = []
fiber = Fiber.new { "Hello from fiber" }

puts " Main Thread: #{main_thread}"
puts "Additional Threads: #{additional_threads}"
puts "Fiber: #[fiber}"

3.times do |i|
additional_threads << Thread.new(i) do |index|
sleep(1)
puts "Additional Thread #{index + 1}: #{Thread.current}"
end
end

puts "Fiber value: #{fiber.resume}"

additional_threads.each(&:join)

puts "All threads have completed."
```

```
C:\Users\student.PROGRAMMING404\Desktop\21MIS1021 VIMAL KUMAR S>ruby 3.rb
 Main Thread: #<Thread:0x0000020eaf7aa070 run>
Additional Threads: []
Fiber: #[fiber}
Fiber value: Hello from fiber
Additional Thread 1: #<Thread:0x0000020eb5c64f60 3.rb:10 run>
Additional Thread 2: #<Thread:0x0000020eb5c64e48 3.rb:10 run>
Additional Thread 3: #<Thread:0x0000020eb5c64d58 3.rb:10 run>
All threads have completed.
```

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

```
File    Edit    View

fiber = Fiber.new do |outer_variable|
puts "Fiber started. Outer variable: #{outer_variable}"

Thread.new do
inner_variable = "Inner value"
puts "Inner thread. Inner variable: #{inner_variable}. Outer variable: #{outer_variable}"
end.join

puts "Fiber resumed. Outer variable: #{outer_variable}"
end

outer_variable = "Outer Value"
fiber.resume(outer_variable)

puts"Program successfully katham katham."
```

OUTPUT:

```
C:\Users\student.PROGRAMMING404\Desktop\21MIS1021 VIMAL KUMAR S>ruby 4.rb
Fiber started. Outer variable: Outer Value
Inner thread. Inner variable: Inner value. Outer variable: Outer Value
Fiber resumed. Outer variable: Outer Value
Program successfully katham katham.

C:\Users\student.PROGRAMMING404\Desktop\21MIS1021 VIMAL KUMAR S>
```

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

CODE:

```
outer_variable = "Outer value"

outer_thread = Thread.new(outer_variable) do |outer_var|
  puts "Outer thread started. Outer variable: #{outer_var}"

  outer_fiber = Fiber.new do |inner_variable|
    puts "Inner fiber. Inner variable: #{inner_variable}. Outer variable: #{outer_var}"
  end

  inner_variable = "Inner value"
  outer_fiber.resume(inner_variable)

  puts "Outer thread resumed. Outer variable: #{outer_var}"
end

outer_thread.join

puts "Program finished."
```

OUTPUT:

```
C:\Users\Dell\Desktop\21MIS1021 VIMAL KUMAR S>ruby 5.rb
Outer thread started. Outer variable: Outer value
Inner fiber. Inner variable: Inner value. Outer variable: Outer value
Outer thread resumed. Outer variable: Outer value
Program finished.
```

6. Multi Thread sharing same variable address space.

CODE:

```
File    Edit    View

counter = 0
mutex = Mutex.new

threads = []

20.times do
  threads << Thread.new do
    mutex.synchronize do
      counter += 1
    end
  end
end

threads.each(&:join)

puts "Counter value: #{counter}"
```

OUTPUT:

```
C:\Users\Dell\Desktop\21MIS1021 VIMAL KUMAR S>ruby 6.rb
Counter value: 20

C:\Users\Dell\Desktop\21MIS1021 VIMAL KUMAR S>
```

7. Write a separate program using the following functions:
    a. Thread – Stop and Run
    b. Thread – Wakeup
    c. Thread – Value
    d. Thread – Pass
    e. Thread –Priority
    f. Thread – Mutex
    g. Thread – Fork

CODE:

File Edit View

```
# Thread - Stop and Run
thread1 = Thread.new do
  puts "Thread 1 is running..."
  Thread.stop
  puts "Thread 1 is resumed!"
end

# Thread - Wakeup
thread2 = Thread.new do
  puts "Thread 2 is sleeping..."
  sleep 2
  thread1.wakeup
end

thread1.run
thread2.join

# Thread - Value
thread3 = Thread.new do
  result = 10 + 20
  Thread.current[:result] = result
end

thread3.join
puts "Thread 3 value: #{thread3[:result]}"

# Thread - Pass
print "Enter a value to pass to Thread 4: "
value = gets.chomp.to_i

thread4 = Thread.new(value) do |passed_value|
  puts "Thread 4 received value: #{passed_value}"
end
```

```
thread4.join

# Thread - Priority
thread5 = Thread.new do
  puts "Thread 5 is running with normal priority"
end

thread6 = Thread.new do
  puts "Thread 6 is running with high priority"
end

thread5.priority = 0
thread6.priority = 2

thread5.join
thread6.join

# Thread - Mutex
counter = 0
mutex = Mutex.new

threads = []

10.times do
  threads << Thread.new do
    mutex.synchronize do
      counter += 1
    end
  end
end

threads.each(&:join)
puts "Counter value: #{counter}"
```

```
# Thread - Fork
if Process.respond_to?(:fork)
  puts "Parent process ID: #{Process.pid}"
  child_pid = fork do
    puts "Child process ID: #{Process.pid}"
  end

  Process.wait(child_pid)
end
ss
```

OUTPUT:

```
C:\Users\Dell\Desktop\21MIS1021 VIMAL KUMAR S>ruby 7.rb
Thread 2 is sleeping...
Thread 1 is running...
Thread 1 is resumed!
Thread 3 value: 30
Enter a value to pass to Thread 4: 1021
Thread 4 received value: 1021
Thread 6 is running with high priority
Thread 5 is running with normal priority
Counter value: 10
```