Universidad de La Laguna
E.S. Ingeniería y Tecnología
Sección: Ingeniería Informática
Área: Lenguajes y Sistemas Informáticos

Lenguajes y Paradigmas de Programación
Hoja de problemas Tema #5

1. ¿Qué es una fibra en Ruby?

2. ¿Qué se obtiene como salida? Describa el comportamiento del programa.

```
words = Fiber.new do
  File.foreach("fichero") do |line|
    line.scan(/\w+/) do |word|
      Fiber.yield word.downcase
    end
  end
end
contadores = Hash.new(0)
while word = words.resume
  contadores[word] += 1
end
contadores.keys.sort.each {|k| print "#{k}:#{contadores[k]} "}
```

3. ¿Qué se obtiene como salida? Describa el comportamiento del programa.

```
twos = Fiber.new do
  num = 2
  loop do
    Fiber.yield(num) unless num % 3 == 0
    num += 2
  end
end
10.times {print twos.resume, " "}
```

4. ¿Qué es un `Thread`?

5. ¿Cómo se crean los *Thread* en Ruby?

6. ¿Qué métodos proporciona Ruby para la manipulación de *Threads*?

7. ¿Qué se obtiene como salida? Describa el comportamiento del programa.

```
print Thread.main
print "\n"
t1 = Thread.new {sleep 100}
Thread.list.each {|thr| p thr }
print "Current thread = " + Thread.current.to_s
print "\n"
t2 = Thread.new {sleep 100}
Thread.list.each {|thr| p thr }
print Thread.current
print "\n"
Thread.kill(t1)
Thread.pass                          # pass execution to t2 now
t3 = Thread.new do
  sleep 20
  Thread.exit                        # exit the thread
end
Thread.kill(t2)                      # now kill t2
Thread.list.each {|thr| print thr }
# now exit the main thread (killing any others)
Thread.exit
```

8. Describa el tipo de acceso que tienen los *Threads* a los distintos tipos de variables en Ruby.

9. ¿Cómo se tratan las variables locales a los *Threads* en Ruby?

10. ¿Qué se obtiene como salida? Describa el comportamiento del programa.

```
count = 0
threads = []
10.times do |i|
  threads[i] = Thread.new do
    sleep(rand(0.1))
    Thread.current["mycount"] = count
    count += 1
  end
end
threads.each {|t| t.join; print t["mycount"], ", "}
puts "count = #{count}"
```

11. ¿Qué es una condición de carrera (*race condition*)?

12. ¿Qué se obtiene como salida? Describa el comportamiento del programa.

```
def inc(n)
  n + 1
end

sum = 0
threads = (1..10).map do
   Thread.new do
     10_000.times do
        sum = inc(sum)
     end
   end
end

threads.each(&:join)
print sum
```

13. ¿Qué es la exclusión mutua (*mutual exclusión*)?

14. ¿Para qué se utilizan los objetos de la clase `Mutex` en Ruby?

15. ¿Qué se obtiene como salida? Describa el comportamiento del programa.

```
def inc(n)
  n + 1
end

mutex = Mutex.new

sum = 0
threads = (1..10).map do
   Thread.new do
     10_000.times do
        mutex.synchronize do
          sum = inc(sum)
        end
     end
   end
end

threads.each(&:join)
print sum
```

16. ¿Qué es un abrazo mortal (*deadlock*)?

17. ¿Qué mecanismo proporciona Ruby para evitar los *deadlock*?

18. ¿Qué se obtiene como salida? Describa el comportamiento del programa.

```
mutex = Mutex.new
cv = ConditionVariable.new

a = Thread.new {
   mutex.synchronize {
      print "A: Esta en una region critica, esperando por cv\n"
      cv.wait(mutex)
      print "A: Esta en la region critica de nuevo!. Sigue\n"
   }
}

print "En medio\n"

b = Thread.new {
   mutex.synchronize {
      puts "B: Esta en la region critica pero tiene a cv"
      cv.signal
      puts "B: Esta en la region critica, Saliendo"
   }
}

a.join
b.join
```