**Course:** ENSF 607–Fall 2020

**Lab #:** Lab 6

**Student Name:** Patrick Kwan

**Submission Date:** November 9, 2020

# Introduction

This lab report analyzes the results of Laboratory 6 of ENSF 607 which covers the topic of clients and servers. This lab report will be composed of three separate sections each showing the results of laboratory exercise 1, 2 and 3.

# Results

## Exercise 1 – Palindrome Server

### Task 1 & 2 Synchronization

In task 1 the program outputs numbers representing a counter. As the two threads are not synchronous the same number can be outputted twice. There needs to be readjustments such that the two threads are synchronous to each other.

Sample Output before modifications:

0

1

2

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

In the sample above the number 2 is repeated twice. In order to fix this both increment in resource and run() in simple thread needs to have the keyword synchronized attach to it. This is the result after this modification.

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

No numbers are repeated twice because each thread needs to complete its operations before another thread can start.

## Task 3 Runnable Interface

In task 3 the simple thread will not inherit from thread, rather it will utilize the runnable interface. In order to do this, extends Thread needs to be changed to implements Runnable. Then two SimpleThread objects of SimpleThread type must be created and those objects need to be passed Thread Constructors that create two thread objects. The following is the results of the modified code.

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

## Exercise 2

### Task 1 Randomized Number

In task 1 five threads are generated, each storing a random value from 0-100. In this implementation the resource class has been readjusted to store one random number in the member randNum. In the ThreadApp five runnable objects are initiated and stored in a List for useful reference later in the program. Those runnable objects are then used to initiate 5 separate threads which are also stored in its own list. At the end of the program using static functions in the app, the sum of all random numbers are calculated by incrementing through the list storing the runnable objects. This sum along with the information of each runnable object is outputted to the console as seen below.

Five Threads 1 to 5 Started:

Thread1 has now started and has a random number of: 45

Thread1 has now ended.

Thread2 has now started and has a random number of: 88

Thread2 has now ended.

Thread3 has now started and has a random number of: 86

Thread3 has now ended.

Thread4 has now started and has a random number of: 3

Thread4 has now ended.

Thread5 has now started and has a random number of: 19

Thread5 has now ended.

Are threads Alive: false

The sum of all values are: 241

Print out all simple threads.

Thread: Thread1

Value Stored: 45

Thread: Thread2

Value Stored: 88

Thread: Thread3

Value Stored: 86

Thread: Thread4

Value Stored: 3

Thread: Thread5

Value Stored: 19

Note that the sum matches expectation of 241 based on the values outputted by the program.

### Task 2 Randomized Number

In task 2 five threads are generated each storing a random value from 0-100 in a shared collection. In this implementation, the resource class not used exercise 1 is brought back and modified to have a randomStorage vector to store this data as oppose to the original counter variable. The resource class has two main methods: an add method that uses collection methods to add a new random number inputted when a new thread is created; and a calcSum method that iterates through the vector and returns the sum of all random values. An aggregation of these values using calcSum is outputted at the end of the program. Below is the results.

Threads 1 to 5 Started:

Added in 12

Added in 93

Added in 7

Added in 60

Added in 55

All threads completed.

The final score is: 227.

Values stored in Collection is:

Data: 12

Data: 93

Data: 7

Data: 60

Data: 55

The program outputs the correct sum of all values stored in collection.

### Task 3 Thread Pool

At this point task 2 and task 1 dealt with threads initiated individually. In this task all the threads will be initiated by using a thread pool. The sum of all the random numbers generated by the thread along with what is stored in the vector collection is outputted. The following is the output of this program.

Added in 61

Added in 59

Added in 34

Added in 22

Added in 58

The sum is: 234

Values stored is:

Data: 58

Data: 59

Data: 61

Data: 22

Data: 34

The sum of 234 is correct and all data added to the resource is properly outputted. This implies that the implementation is correct.