

CS6500 Assignment1

Yijia Ma

1. Repo URL: <https://github.com/CocoCSforever/cs6650/tree/main>
 - 1.1. Please see SkierServlet API in Assignment1_Servlet.
 - 1.2. Please see Client API in Assignment1_Clients(I changed the name of the project lately which may show some warning of the dependencies but it runs well.):
Client Part1 in Assignment1_Clients/src/main/java/clients/client1.
Client Part2 in Assignment1_Clients/src/main/java/clients/client2.
Client Test for single thread in Assignment1_Clients/src/test/SkiersApiTest
2. Servlet Design(/skiers)
 - 2.1. doPost
 - a. Sample request url:
`/ {resortID}/seasons/{seasonID}/days/{dayID}/skiers/{skierID}`
 - b. Validation Steps
 - i. The function `isValid` serves as a basic validation. It verifies whether the `urlPath` contains the seven parts as outlined in our sample request URL and ensures that the second, fourth, and sixth parts are exactly equal to "seasons," "days," and "skiers," respectively.
 - ii. A URL path that passes the validation indicates that it has the same structure as outlined in the sample request but is not necessarily guaranteed to have a valid value for a `LiftRide` object.
 - c. Status Code
 - i. 404 NOT_FOUND
 1. If the `urlPath` is null or empty, it should return a response with a 404 status code and write a message into `req.body` stating "missing parameters" along with the `urlPath`.
 - ii. 400 BAD_REQUEST
 1. if `isValid(urlPath)` returns false, it should return a response with a 400 status code and write a message into `req.body` stating "Invalid URL Path parameters: " along with the `urlPath`.
 2. If the parameters interpreted from the URL are not valid as attributes in a `LiftRide` object, it should return a response with a 400 status code and write a message into `req.body` stating "Bad URL path parameters: " along with the `urlPath` and the error message obtained from `e.getMessage()`.
 - iii. 201 CREATED
 1. If a `LiftRide` object is successfully created without any errors, it should return a response with a 201 status code and write the string representation of the object into `req.body`.
3. Client Design
 - 3.1. Important instruction to reproduce my result
To reproduce my result, please change `Client1_Multi_Threads`, line 8 with the desired threads and `requestsPerThreads`, and `Client1ProducerConsumer`,

line 8 to align the size of thread pool with the number of desired threads(it should not be less than 32).

Please change Client2_Multi_Threads, line 17 with the desired threads and requestsPerThreads, and Client1ProducerConsumer, line 12 to align the size of thread pool with the number of desired threads(it should not be less than 32).

3.2. Class structure

My class design for Part1 and Part2 follows the same pattern. Take Client Part1 for example, it includes:

- a. Client1_Multi_Threads
 - i. This is where our main method resides. It calls a helper function(pass the number of threads and requests per threads) and records wall time and other necessary information, and prints them out.
- b. Client1ThreadHelper
 - i. Fixed thread pool of size 1 is used for generating ready-to-send liftRide events since we are required to use a single dedicated thread.
 - ii. client1StartNThreads is the helper function which returns the wall time for sending client requests. It includes:
 1. Create a ProducerConsumer object.
 2. Set required CountdownLatch.
 3. Generate LiftRide events.
 4. send client requests using the generated events.
 5. wait until all client requests are successfully sent and get good responses.
 6. record wall time and return.
 - iii. generateLiftRideData submit the produce() task to the thread pool and wait until all events are ready using latch.await().
 - iv. sendClientRequest submit the consume() task to thread pool.
- c. Client1ProducerConsumer
 - i. Constructor takes nOfThread as the number of threads or clients, and requestPerThread as the number of requests to send per thread.
 - ii. Fixed thread pool of size \$nOfThread is only used for running client1Runnable(\$ represents the value of the variable).
 - iii. produce() will create \$nOfThread client1Runnable and generate \$requestPerThread liftRide events for each client1Runnable. After the data for a client is ready to go, it adds the client1Runnable to the taskQueue.
 - iv. consume() will take \$nOfThread client1Runnable from the taskQueue one by one and submit the runnable object to the thread pool.
- d. Client1Runnable
 - i. This is a class that implements Runnable Interface and run() defines specific actions for each thread to execute.
 - ii. Depending on requirements, I set some static variables for class Client1Runnable, and these variables can be shared by all threads created based on Client1Runnable instance, eg. successCounter, failCounter, CountdownLatch, BufferedWriter.

- iii. Each Runnable instance has their own SkierAPI created by different ApiClient so that each thread represents a respective client and sends its requests for the client.
- iv. ArrayList<LiftRideInfo> is a list of size \$requestPerThread and contains liftRide events generated by produce().

Some adds on for Client2:

- a. Client2Runnable
 - i. To write the required info for each request, I created a writeToFile function which is called when we get a successful response. Since writeToFile is a logically synchronized function, I created a fixed thread pool of size 1 in class Client2Runnable to submit a write task to it.
 - b. Client2_calculate
 - i. It's created for analyzing the data we wrote to output.csv such as to calculate the mean response time, median response time and so on.
- 3.3. Packages
- a. My packages(src/main/java/clients)
 - i. I write Client Part1 and Part2 in two folders in package clients.
 - ii. I modified SkiersApiTest: added one test for each client part.
 - b. Swagger generated API(src/main/java/io.swagger.client)
 - i. I mainly relied on ApiClient, ApiCallback, SkiersApi to send client requests to my server.
 - ii. I added a constructor for SkiersApi that takes an ApiClient instance so that every thread can have a different client instance instead of all set to the configuration's default client.
 - iii. I implement ApiCallback to have different tasks done onFailure and onSuccess of client requests.
4. Client (Part 1) - This should be a screenshot of your output window with your wall time and throughput. Also make sure you include the client configuration in terms of the number of threads used.
- 4.1. Client configuration
- a. It first created 32 threads each sending 1k POST requests. After completing all these requests, it created x threads(x = 32, 80, 120 in different test cases) until 200k POST requests have been sent.
 - b. Please see 4.2 Output window a and f as the best throughput.
 - c. Please see 4.2 Output window g as testing for single thread 10000 requests for single request latency.
- 4.2. Output window
- d. Running Locally(**Best Throughput**)
 - 32 threads * 1k requests
 - 32 threads * 5.25k requests

```

/Users/mayijia/Library/Java/JavaVirtualMachines/corretto-17.0.6/Contents/Home/bin/java ...
1. number of successful requests sent: 200000
2. number of unsuccessful requests (should be 0): 0
3. total run time (wall time) for all phases to complete: 43 seconds
4. total throughput in requests per second : 4555.497346422796
  
```

- e. Running Locally
32 threads * 1k requests
80 threads * 2.1k requests

```
Client1_Multi_Threads x
/Users/mayijia/Library/Java/JavaVirtualMachines/corretto-17.0.6/Contents/Home/bin/
1. number of successful requests sent: 200000
2. number of unsuccessful requests (should be 0): 0
3. total run time (wall time) for all phases to complete: 54 seconds
4. total throughput in requests per second : 3641.130206816196
```

- f. Running Locally
32 threads * 1k requests
120 threads * 1.4k requests

```
Client1_Multi_Threads x
/Users/mayijia/Library/Java/JavaVirtualMachines/corretto-17.0.6/Contents/Home/bin/
1. number of successful requests sent: 200000
2. number of unsuccessful requests (should be 0): 0
3. total run time (wall time) for all phases to complete: 50 seconds
4. total throughput in requests per second : 3964.0853863992234
```

- g. Running on EC2
32 threads * 1k requests
32 threads * 5.25k requests

```
clients.client2.Client2_Multi_Threads x Client1_Multi_Threads x
/Users/mayijia/Library/Java/JavaVirtualMachines/corretto-17.0.6/Contents/Home/bin/java ...
1. number of successful requests sent: 200000
2. number of unsuccessful requests (should be 0): 0
3. total run time (wall time) for all phases to complete: 600 seconds
4. total throughput in requests per second : 332.9221744478902

Process finished with exit code 0
```

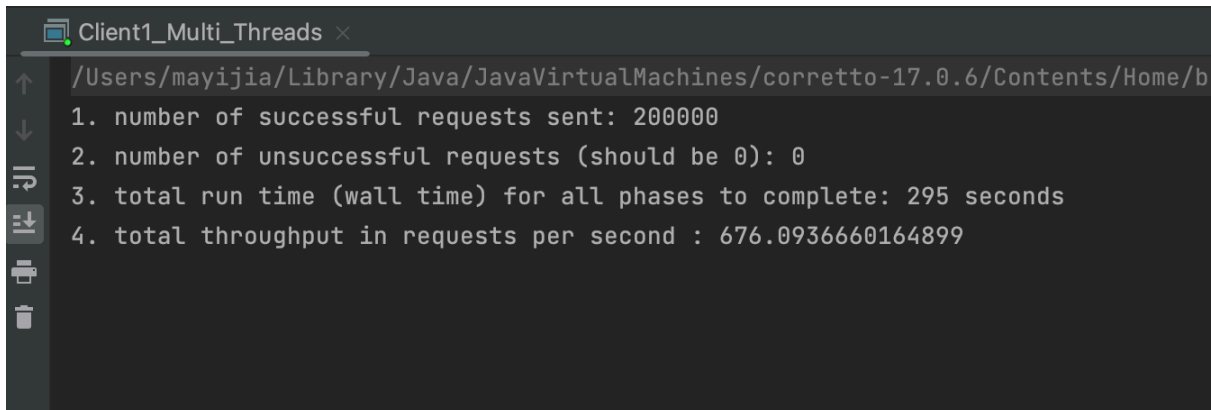
- h. Running on EC2
32 threads * 1k requests
80 threads * 2.1k requests

```
Client1_Multi_Threads x
/Users/mayijia/Library/Java/JavaVirtualMachines/corretto-17.0.6/Contents/Home/bin/
1. number of successful requests sent: 200000
2. number of unsuccessful requests (should be 0): 0
3. total run time (wall time) for all phases to complete: 314 seconds
4. total throughput in requests per second : 635.7844944877485
```

i. Running on EC2(**Best Throughput**)

32 threads * 1k requests

120 threads * 1.4k requests



```
Client1_Multi_Threads x
/Users/mayijia/Library/Java/JavaVirtualMachines/corretto-17.0.6/Contents/Home/b
1. number of successful requests sent: 200000
2. number of unsuccessful requests (should be 0): 0
3. total run time (wall time) for all phases to complete: 295 seconds
4. total throughput in requests per second : 676.0936660164899
```

j. Single Thread sending 10000 requests



```
@Test
public void clientSingleRequestTimeTest() throws Exception {
    long startTime = System.currentTimeMillis();
    clientStartWithThreads(4096, 1, 10000);
    System.out.println("total run time (wall time) for 10000 request/thread to complete: " + 1.0*(System.currentTimeMillis()-startTime)/1000 + " seconds");
    System.out.println("run time (wall time) per request: " + 1.0*(System.currentTimeMillis()-startTime)/ ClientRunnable.getSuccessCounter() + " milliseconds per request");
}

Run: SkiersApiTest.clientSingleRequestTimeTest
Tests passed: 1 of 1 test - 18 sec 242 ms
/Users/mayijia/Library/Java/JavaVirtualMachines/corretto-17.0.6/Contents/Home/bin/java ...
total run time (wall time) for 10000 request/thread to complete: 18.183 seconds
run time (wall time) per request: 1.8184 milliseconds per request
Process finished with exit code 0
```

4.3. Observation

- a. Optimal throughput is achieved with 32 client threads locally but with 120 client threads on EC2. This difference is likely because local runs share resources between server and client threads, limiting the total available, while EC2 provides more resources, allowing for more client threads to operate without depleting the system's thread capacity, thereby supporting increased throughput.

4.4. Little's Law

a. Observation

We'll use 32 client threads to analyze it since this case maintains a constant long-term average number of clients. The long term average throughput is 32 threads(my server max thread is set to default 200). Response time for each request: 1.82 millisecond, thus the arrival/exit rate should be $32/1.82 = 17.58$ per millisecond. However, it turns out to take 4.5 requests per millisecond.

b. Justification

The huge gap may be caused by the synchronized functions that I used to increment and get successCounter and failCounter which makes the threads to wait for each other until the lock on the instance is released. In addition, when we use one single thread to send 10000 requests, we are using one client and possibly using the same connection to the server while in a multithread case, we spend more time establishing a connection.

5. Client (Part 2) - run the client as per Part 1, showing the output window for each run with the specified performance statistics listed at the end.

5.1. Client configuration

- a. It first created 32 threads each sending 1k POST requests. After completing all these requests, it created x threads (x = 32, 80, 120 in different test cases) until 200k POST requests have been sent.
 - b. Please see 5.2 Output window e and i as the best throughput.
 - c. Please see 5.2 Output window j as testing single thread 10000 requests for single request latency.
- 5.2. Output window
- a. Running on EC2
 - 32 threads * 1k requests
 - 32 threads * 5.25k requests
 (with extra 1 threads writing data to file and 1 thread generating liftRide Event)

```
clients.client2.Client2_Multi_Threads x Client2_calculate x
/Users/mayijia/Library/Java/JavaVirtualMachines/corretto-17.0.6/Contents/Home/bin/java ..
1. number of successful requests sent: 200000
2. number of unsuccessful requests (should be 0): 0
3. total run time (wall time) for all phases to complete: 605.185 seconds
4. total throughput in requests per second : 330.4774573064435
```

```
clients.client2.Client2_Multi_Threads x Client2_calculate x
/Users/mayijia/Library/Java/JavaVirtualMachines/corretto-17.0.6/Contents/Home/bin/java ..
mean response time (milliseconds): 94
median response time (milliseconds): 90
throughput = total number of requests/wall time (requests/second): 330.4774573064435
p99 (99th percentile) response time: 191
min response time (milliseconds): 76
max response time (milliseconds): 819

Process finished with exit code 0
```

- b. Running on EC2
 - 32 threads * 1k requests
 - 60 threads * 2.8k requests
 (with extra 32 threads writing data to file and 1 thread generating liftRide Event)

```
clients.client2.Client2_Multi_Threads x Client2_calculate x
/Users/mayijia/Library/Java/JavaVirtualMachines/corretto-17.0.6/Contents/Home/bin/java ..
1. number of successful requests sent: 200000
2. number of unsuccessful requests (should be 0): 0
3. total run time (wall time) for all phases to complete: 378.003 seconds
4. total throughput in requests per second : 529.0963299233075
```

```
clients.client2.Client2_Multi_Threads x Client2_calculate x
/Users/mayijia/Library/Java/JavaVirtualMachines/corretto-17.0.6/Contents/Home/bin/java
mean response time (milliseconds): 98
median response time (milliseconds): 89
throughput = total number of requests/wall time (requests/second): 529.0963299233075
p99 (99th percentile) response time: 193
min response time (milliseconds): 75
max response time (milliseconds): 647

Process finished with exit code 0
```

c. Running on EC2

32 threads * 1k requests

60 threads * 2.8k requests

(with extra 1 thread writing data to file and 1 thread generating liftRide Event)

Compared to b, The number of threads writing data to file(output.csv) doesn't make much difference on throughput.

```
Client2ThreadHelper
> clientdraft
> io.swagger.client
AndroidManifest.xml
test
java
  io.swagger.client.api
    ResortsApiTest
    SkiersApiTest
    StatisticsApiTest
.travis.yml

15
16
17
18
19
20
21
22
23
24
25
26
27
long wallTime1 = Client2ThreadHelper.client2StartNThreads( nOfThreads: 32, requestPerThread: 1000, bf);
long wallTime2 = Client2ThreadHelper.client2StartNThreads( nOfThreads: 60, requestPerThread: 2800, bf);
long time = wallTime1 + wallTime2;

Client2ThreadHelper.executor.shutdown();
Client2ProducerConsumer.executor.shutdown();
Client2Runnable.executor.shutdown();
bf.write( "all: "+time);
System.out.println("1. number of successful requests sent: " + Client2Runnable.getSuccessCounter());
System.out.println("2. number of unsuccessful requests (should be 0): " + Client2Runnable.getFailCounter());
System.out.println("3. total run time (wall time) for all phases to complete: " + 1.0*time/1000 + " seconds");
System.out.println("4. total throughput in requests per second : " + 1.0* Client2Runnable.getSuccessCounter()/(time*1000));

run: clients.client2.Client2_Multi_Threads x Client2_calculate x
/Users/mayijia/Library/Java/JavaVirtualMachines/corretto-17.0.6/Contents/Home/bin/java ...
1. number of successful requests sent: 200000
2. number of unsuccessful requests (should be 0): 0
3. total run time (wall time) for all phases to complete: 389.96 seconds
4. total throughput in requests per second : 512.8731151913016
```

```
clients.client2.Client2_Multi_Threads x Client2_calculate x
/Users/mayijia/Library/Java/JavaVirtualMachines/corretto-17.0.6/Contents/Home/bin/java ...
mean response time (milliseconds): 100
median response time (milliseconds): 91
throughput = total number of requests/wall time (requests/second): 512.8731151913016
p99 (99th percentile) response time: 202
min response time (milliseconds): 75
max response time (milliseconds): 579

Process finished with exit code 0
```

d. Running on EC2

32 threads * 1k requests

80 threads * 2.8k requests

(with extra 1 thread writing data to file and 1 thread generating liftRide Event)

```
n: clients.client2.Client2_Multi_Threads x Client2_calculate x
/Users/mayijia/Library/Java/JavaVirtualMachines/corretto-17.0.6/Contents/Home/bin/java ...
1. number of successful requests sent: 200000
2. number of unsuccessful requests (should be 0): 0
3. total run time (wall time) for all phases to complete: 327.894 seconds
4. total throughput in requests per second : 609.9532165882877

Process finished with exit code 0
```

```
clients.client2.Client2_Multi_Threads x Client2_calculate x
/Users/mayijia/Library/Java/JavaVirtualMachines/corretto-17.0.6/Contents/Home/bin/java ...
mean response time (milliseconds): 102
median response time (milliseconds): 91
throughput = total number of requests/wall time (requests/second): 609.9532165882877
p99 (99th percentile) response time: 203
min response time (milliseconds): 75
max response time (milliseconds): 687
```

e. Running on EC2(**Best Throughput**)

32 threads * 1k requests

120 threads * 1.4k requests

(with extra 1 thread writing data to file and 1 thread generating liftRide Event)

```
n: clients.client2.Client2_Multi_Threads x Client2_calculate x
/Users/mayijia/Library/Java/JavaVirtualMachines/corretto-17.0.6/Contents/Home/bin/java ...
1. number of successful requests sent: 200000
2. number of unsuccessful requests (should be 0): 0
3. total run time (wall time) for all phases to complete: 271.597 seconds
4. total throughput in requests per second : 736.3851588935077
```

```
clients.client2.Client2_Multi_Threads x Client2_calculate x
/Users/mayijia/Library/Java/JavaVirtualMachines/corretto-17.0.6/Contents/Home/bin/java ...
mean response time (milliseconds): 112
median response time (milliseconds): 94
throughput = total number of requests/wall time (requests/second): 736.3851588935077
p99 (99th percentile) response time: 253
min response time (milliseconds): 75
max response time (milliseconds): 1357

Process finished with exit code 0
```

f. Running on EC2

Failed with Connection Error when trying to create 150 client threads.

32 threads * 1k requests

150 threads * 2.8k requests

(with extra 1 thread writing data to file and 1 thread generating liftRide Event)

g. Running locally

32 threads * 1k requests

32 threads * 5.25k requests


```
clients.client2.Client2_Multi_Threads x Client2_calculate x
/Users/mayijia/Library/Java/JavaVirtualMachines/corretto-17.0.6/Contents/Home/bin/java ..
1. number of successful requests sent: 200000
2. number of unsuccessful requests (should be 0): 0
3. total run time (wall time) for all phases to complete: 51.543 seconds
4. total throughput in requests per second : 3880.2553208001086

Process finished with exit code 0
```

```
clients.client2.Client2_Multi_Threads x Client2_calculate x
/Users/mayijia/Library/Java/JavaVirtualMachines/corretto-17.0.6/Contents/Home/bin/java ..
mean response time (milliseconds): 7
median response time (milliseconds): 3
throughput = total number of requests/wall time (requests/second): 3880.2553208001086
p99 (99th percentile) response time: 116
min response time (milliseconds): 0
max response time (milliseconds): 382
```

- h. Running locally
 - 32 threads * 1k requests
 - 60 threads * 2.8k requests

```
clients.client2.Client2_Multi_Threads x
/Users/mayijia/Library/Java/JavaVirtualMachines/corretto-17.0.6/Contents/Home/bin/java ..
1. number of successful requests sent: 256000
2. number of unsuccessful requests (should be 0): 0
3. total run time (wall time) for all phases to complete: 59.291 seconds
4. total throughput in requests per second : 4317.687338719199
```

```
clients.client2.Client2_Multi_Threads x Client2_calculate x
/Users/mayijia/Library/Java/JavaVirtualMachines/corretto-17.0.6/Contents/Home/bin/java ..
mean response time (milliseconds): 15
median response time (milliseconds): 5
throughput = total number of requests/wall time (requests/second): 4317.687338719199
p99 (99th percentile) response time: 114
min response time (milliseconds): 0
max response time (milliseconds): 10179

Process finished with exit code 0
```

- i. Running locally(**Best Throughput**)
 - 32 threads * 1k requests
 - 120 threads * 1.4k requests

```
clients.client2.Client2_Multi_Threads x Client2_calculate x
/Users/mayijia/Library/Java/JavaVirtualMachines/corretto-17.0.6/Contents/Home/bin
1. number of successful requests sent: 200000
2. number of unsuccessful requests (should be 0): 0
3. total run time (wall time) for all phases to complete: 35.611 seconds
4. total throughput in requests per second : 5616.242172362472

Run: clients.client2.Client2_Multi_Threads x Client2_calculate x
/Users/mayijia/Library/Java/JavaVirtualMachines/corretto-17.0.6/Contents/Home/bin/java ...
mean response time (milliseconds): 9
median response time (milliseconds): 4
throughput = total number of requests/wall time (requests/second): 5616.242172362472
p99 (99th percentile) response time: 112
min response time (milliseconds): 0
max response time (milliseconds): 397

Process finished with exit code 0
```

j. Single Thread sending 10000 requests

```
2_Multi_Threads x SkiersApiTest.client2SingleRequestTimeTest x
✓ Tests passed: 1 of 1 test – 20 sec 115 ms
/Users/mayijia/Library/Java/JavaVirtualMachines/corretto-17.0.6/Contents/Home/bin
total run time (wall time) for 10000 request/thread to complete: 20.043 seconds
run time (wall time) per request: 2.0043 milliseconds per request

Process finished with exit code 0
```

5.3. Little's Law

a. Observation

We'll use 32 client threads to analyze it since this case maintains a constant long-term average number of clients. The min number of threads = 32, this is the long term average requests present in the system. Following the little's law, the arrival rate = $32/2.0 = 16$ requests per millisecond. However, the actual arrival rate (total throughput in request per second) we observed is 3.88 per millisecond.

b. Justification

The significant delay may result from the synchronized functions implemented to increment and retrieve successCounter and failCounter. These synchronized functions introduce a lock on the instance, forcing threads to wait for each other until the lock is released.

In a single-threaded scenario, the client requests are sent sequentially, and there is no contention for shared data. However, in a multi-threaded environment, when multiple threads need access to shared data, the

synchronized functions impose a lock that allows only one thread to access the data at a time. Consequently, every other thread must wait until the current thread exits the method, leading to considerable time wastage. In addition, when we use a single thread to send 10000 requests, we are using one client and possibly using the same connection to the server while in a multithread case, we spend more time establishing a connection.

c. Data processing

```
Client1_Multi_Threads x Client2_Multi_Threads x Client2_calculate x
/Users/mayijia/Library/Java/JavaVirtualMachines/corretto-17.0.6/Contents/Home/bin/java ...
mean response time (millisecs): 19
median response time (millisecs): 6
throughput = total number of requests/wall time (requests/second): 0
p99 (99th percentile) response time: 189
min response time (millisecs): 0
max response time (millisecs): 913

Process finished with exit code 0
```

I created a class Client2_calculate to do the data processing after we got all the information written to the output.csv. Detailed output is included as part of the output window.

6. Single Thread sending 10000 requests

6.1. client1

```
AndroidManifest.xml
test
  java
    io.swagger.client.api
      SkiersApiTest
      SkiersApiTest
      StatisticsApiTest
Run: SkiersApiTest.client1SingleRequestTimeTest
Tests passed: 1 of 1 test - 18 sec 242 ms
total run time (wall time) for 10000 request/thread to complete: 18.183 seconds
run time (wall time) per request: 1.8184 milliseconds per request
Process finished with exit code 0
```

6.2. client2

```
Client2_Multi_Threads x SkiersApiTest.client2SingleRequestTimeTest x
Tests passed: 1 of 1 test - 20 sec 115 ms
/Users/mayijia/Library/Java/JavaVirtualMachines/corretto-17.0.6/Contents/Home/bin/...
total run time (wall time) for 10000 request/thread to complete: 20.043 seconds
run time (wall time) per request: 2.0043 milliseconds per request

Process finished with exit code 0
```