Faculty of Computers, Informatics and Microelectronics

Technical University of Moldova

PR

Laboratory work #2-3

Laboratory description. HTTP Client with concurrency superpowers.

Authors:
Tanașciuc Macarie

Supervisor:
Alex Gavrișco

Chișinău, 2018

1 Tasks

- Study OSI model, HTTP and implement a client app which can do multiple HTTP requests concurrently.
- Prerequisites
- Read about OSI model (definition, basic info)
- Read about (de)serialization
- Concurrency (definition, primitives etc)

1.1 Metrics Aggregator

Your new client is a super secret organization which needs to collect a bunch of different metrics from an object and aggregate them. An object is a real location that has some devices which collects data from sensors. Each device has an id, type or sensor type and a value which describes device's state. So, metrics is just a fancy word for data which describes state of your object from a perspective (e.g. temperature, air pressure etc). However, there are some problems about those devices:

- Each device has its own URL and can be accessed only using a secret key
- Devices can return values in different formats (JSON, XML, CSV)
- Depending on format, a device can return multiple values (e.g. a box which has multiple sensors)
- Sometimes a device can respond you in up to 29 sec.
- And the most important constraint - your secret key is valid only 30 sec. (because the organization is super secret and don't want to leak access keys).

Here's what functionality the app must offer:

- Request your secret key at https://desolate-ravine-43301.herokuapp.com/, in response you'll receive a list of URLs (for each device)
- Using your secret key, request data from all devices concurrently
- If you get an error related to your access key, go back to step 1 and retry
- Parse data from all devices
- Aggregate all responses ordering by sensor type (example of output below)

2 Implementation

In the code bellow i create a thread for each request after what i join the threads so each request would be processed first before doing some other tasks.

```
Tr = [Threads(urls[i], headers) for i in range(len(urls))]
  for i in range(len(urls)):
      Tr[i].start()
  for i in range(len(urls)):
      Tr[i].join()
  #queue processing
  queue.remove(None)
  queue.sort(key=lambda x: (int(x[1]), float(x[2])))
  s = [i for i in range(6)]
  for i in range(len(queue)):
      output(queue[i], s)
```

Each thread processes a request and appends all data downloaded to a queue.

```
class Threads(T):
    def __init__(self, url, headers):
        T.__init__(self)
        self.url, self.headers = url, headers

    def run(self):
        start = time.time()
        response = requests.get(self.url, headers=self.headers)
        roundup = time.time() - start
        print(BC.BOLD, "\n", self.name, "Status Code -",
              response.status_code, "(", round(roundup, 4), "s )\nUrl:", self.url)

        if response.headers["Content-Type"] == "text/csv":
            appending = parsing(response.headers["Content-Type"], response.text)
            for i in range(len(appending)):
                queue.append(appending[i])
        else:
            queue.append(parsing(response.headers["Content-Type"], response.text))
```

Before appending to queue the data is parsed to a general format according to its content type received from headers.

```
def parsing(content_type, text):
    if content_type == "Application/xml":
        soup = BeautifulSoup(text, 'html.parser')
        return [soup.device['id'], soup.type.string, soup.value.string]
    elif content_type == "Application/json":
        j_file = json.loads(text)
        return [j_file['device_id'], j_file['sensor_type'], j_file['value']]
    elif content_type == "text/csv":
        text = [s.split(",") for s in text.splitlines()]
        x = [[text[i][0], text[i][1], text[i][2]] for i in range(len(text)) if i != 0]
        return x
    else:
        print(text)
```

After what the data is outputed according the sensors from queue.

```
def output(block, s):
    device_id, sensor, value = block[0], int(block[1]), block[2]
    if sensor in s:
        if sensor == 0:
            print(BC.OKGREEN, "\nTemperature:")
        if sensor == 1:
            print(BC.HEADER, "\nHumidity:")
        if sensor == 2:
            print(BC.FAIL, "\nAlien Presence:")
        if sensor == 3:
            print(BC.ENDC, "\nDark Matter")
        if sensor == 4:
            print(BC.OKBLUE, "\nGhost Presence:")
        if sensor == 5:
            print(BC.WARNING, "\nDoes this Sensor even exist:")
        s.remove(sensor)
    # ----------------------
    print(" -Device", device_id, "-", value)
    # ----------------------
```

2.1 Result

Figure 2.1– Requests processed concurrently

```
Temperature:
 •Device voiglopbuoped - 0.092970155
 •Device jmkxvbwgvbqzg - 0.097262
 •Device yarkctzkjkziv - 0.865175
 •Device vksjfjzalbtzs - 0.956021

Humidity:
 •Device hsbzrjxawnwek - 0.029671282
 •Device vkurupifvizrg - 0.710744
 •Device vgseycjpjhynu - 0.7698891
 •Device fdzdcekxbakjq - 0.81439453

Alien Presence:
 •Device hhjuvrusqfgqv - 0.30380213

Dark Matter
 •Device ucwksxbgyraom - 0.075037
 •Device ynsgrussvmaoz - 0.716368
 •Device udtrzqmdqiyco - 0.9047762

Ghost Presence:
 •Device ekjyixjrscctn - 0.520380
 •Device omervjarzlntx - 0.887447

Does this Sensor even exist:
 •Device bsbojifqgzsnw - 0.331368
 •Device tmttcoanatyyi - 0.58938307

Process finished with exit code 0
```

Figure 2.2– Metrics Aggregations

3 Conclusions

During this laboratory work i learned to how make requests on api using the http headers and how to operate with threads.

References

1 **Task** - https://github.com/Alexx-G/PR-labs/blob/master/lab2-3.md

2 **Git Repository** - https://github.com/LordOfNightmares/PR-Labs/tree/master/Lab2

3 **python multithreading1** - https://www.toptal.com/python/beginners-guide-to-concurrency-and-parallelism-in-python

4 **python multithreading2** - https://www.tutorialspoint.com/python3/python_multithreading.htm

5 **Status codes** - https://en.wikipedia.org/wiki/List_of_HTTP_status_codes