# COMPUTER SCIENCE AND ENGINEERING

## Indian Institute of Technology, Palakkad

## CS5107: Programming Lab

19 Sep, 2021

***Lab 4: Threads and sockets***

Time: 1 week

Max points: 100

1. Your codes should be compatible with *Python3* and be well-commented. Also, make sure to name your files properly.

2. This assignment will involve *multi-threading* and *socket-programming* in Python. A few resources that can help in this assignment are below:

    1. A thread is a separate flow of execution. This means that your program will have two things happening at once. But for most Python 3 implementations the different threads do not actually execute at the same time: they merely appear to. To know more, visit `https://realpython.com/intro-to-python-threading/`
    2. Python's `threading` module provides higher-level threading interfaces to work with multiple threads. For more details, visit `https://docs.python.org/3/library/threading.html`
    3. Python's `concurrent.futures` module provides a high-level interface for asynchronously executing callables. For more details, visit `https://docs.python.org/3/library/concurrent.futures.html`
    4. Sockets and the socket API are used to send messages across a network. They provide a form of inter-process communication (IPC). The network can be a logical, local network to the computer, or one that's physically connected to an external network, with its own connections to other networks. To know more, visit `https://realpython.com/python-sockets/`
    5. Details of Python's `socket` module is available at `https://docs.python.org/3/library/socket.html`
    6. Python's `time` module provides various time-related functions. For more details, visit `https://docs.python.org/3/library/time.html`
    7. Python's `random` module implements pseudo-random number generators for various distributions. For more details, visit `https://docs.python.org/3/library/time.html`
    8. JSON (JavaScript Object Notation), specified by RFC 7159 (which obsoletes RFC 4627) and by ECMA-404, is a lightweight data interchange format inspired by JavaScript object literal syntax. Python's `json` module provides API to convert Python objects to JSON objects and vice-versa. For more details, visit `https://docs.python.org/3/library/json.html`

3. Write a program that has 2 real-time threads namely: EG and EC threads. These threads exhibit the following behavior: [20]

    1. EG thread generates events, whereas EC thread consumes them.
    2. EG thread generates a new event $\gamma$ seconds after the consumption of an event. For each event, $\gamma$ is a random variable sampled uniformly from the set $\{1, 2, 3, 4, 5\}$ (values in seconds).

Continued on next page...

   3. EG thread generates only one event at a time. If an unconsumed event is present, EG thread waits for EC thread to consume it before generating a new event.
   4. As soon as an event occurs, EC thread starts to consume it. It takes $\mu$ seconds to consume an event. For each event, $\mu$ is a random variable sampled uniformly from the set $\{1, 2, 3, 4, 5\}$ (values in seconds).
   5. EG thread should generate a total of 10 events. This number is an information private to EG thread is not shared with EC thread.
   6. EC thread should print an appropriate message after it finishes consuming all events.

   ***HINT**: Use `Event` object from `threading` module.*

   A sample output when EG thread is configured to generate 5 events:

```
Time 0s: Event scheduled at 1s
Time 1s: Event occurred
Time 2s: Event processed
Time 2s: Event scheduled at 3s
Time 3s: Event occurred
Time 5s: Event processed
Time 5s: Event scheduled at 10s
Time 10s: Event occurred
Time 15s: Event processed
Time 15s: Event scheduled at 16s
Time 16s: Event occurred
Time 18s: Event processed
Time 18s: Event scheduled at 23s
Time 23s: Event occurred
Time 24s: Event processed
All events have been processed
```

4. Write a program that has 5 real-time threads, each corresponding to a person visiting a mall. The people (corresponding threads) should exhibit the following behavior: [20]

   1. Each person reaches the mall at a random amount of time that is uniformly sampled from the set $\{1, 2, \ldots, 19, 20\}$ (values in seconds).
   2. When a person reaches the mall, they wait for others. As soon as everyone arrive, all of them enter the mall.
   3. Subsequently, each of them spend a random amount that uniformly sampled from the set $\{1, 2, \ldots, 9, 10\}$ (values in seconds) of time in the mall and then leaves.

   ***HINT**: Use `Barrier` object from `threading` module.*

   A sample output:

```
Time 5s: Person 3 reached the mall
Time 14s: Person 2 reached the mall
Time 16s: Person 4 reached the mall
Time 18s: Person 1 reached the mall
```

Continued on next page...

```
Time 20s: Person 5 reached the mall
Time 20s: Person 5 enters the mall
Time 20s: Person 3 enters the mall
Time 20s: Person 4 enters the mall
Time 20s: Person 2 enters the mall
Time 20s: Person 1 enters the mall
Time 21s: Person 3 leaves the mall
Time 22s: Person 1 leaves the mall
Time 24s: Person 4 leaves the mall
Time 27s: Person 2 leaves the mall
Time 30s: Person 5 leaves the mall
```

5. Write a program that has 5 real-time threads, each corresponding to a person visiting a shop. People (corresponding threads) should exhibit the following behavior: [20]

    1. Each person reaches the shop at a random amount of time that is uniformly sampled from the set $\{1, 2, 3, 4, 5\}$ (values in seconds).
    2. No more than 2 persons can be in the shop at any time.
    3. As soon a person reaches the shop they enter it. However, if there are 2 people in the shop, he/she has to wait till someone leaves the shop.
    4. After entering, a person spends random amount that is uniformly sampled from the set $\{5, \ldots, 10\}$ (values in seconds) of time in the shop before leaving.

    ***HINT***: *Use `Semaphore` object from `threading` module.*

    A sample output:

    ```
    Time 2s: Person 1 reached the shop
    Time 2s: Person 1 entered the shop
    Time 2s: Person 2 reached the shop
    Time 2s: Person 2 entered the shop
    Time 3s: Person 3 reached the shop
    Time 4s: Person 4 reached the shop
    Time 5s: Person 5 reached the shop
    Time 9s: Person 1 left the shop
    Time 9s: Person 3 entered the shop
    Time 10s: Person 2 left the shop
    Time 10s: Person 4 entered the shop
    Time 15s: Person 3 left the shop
    Time 15s: Person 5 entered the shop
    Time 19s: Person 4 left the shop
    Time 21s: Person 5 left the shop
    ```

6. Create a client and server that do the following: [20]

    - Client creates an array of 10 tuples. Let us denote this array as $\mathcal{A} = [(a_1, b_1), \ldots, (a_{10}, b_{10})]$
    - Client print the array $\mathcal{A}$, and sends it to the server using TCP sockets.

Continued on next page. . .

- Sever creates the array $\mathcal{B} = [(\overline{a}, \overline{b}), (a^{max}, b^{max}), (a^{min}, b^{min})]$ and sends it back to the client using TCP socket. Here, $\overline{a} = \frac{1}{10}\sum_{i=1}^{10} a_i$, $\overline{b} = \frac{1}{10}\sum_{i=1}^{10} b_i$, $a^{max} = \max_{1\leq i\leq 10} a_i$, $b^{max} = \max_{1\leq i\leq 10} b_i$, $a^{min} = \min_{1\leq i\leq 10} a_i$ and $b^{min} = \min_{1\leq i\leq 10} b_i$.
- Client displays the received array $\mathcal{B}$.

***HINT**: Have a look at Python's `json` module.*

7. Create a client and server that do the following: [20]

- Client creates 10 real-time threads; each of them sends a message to the server using TCP socket and waits for a reply.
- Each message received by the sever is sent back to the client after a delay of that is sampled uniformly at random from the set $\{1, 2, 3, 4, 5\}$ (values in seconds).
- Threads display the message received from server. For each thread, the received message should be same as the sent one.

A sample output:

```
Time 0s: Thread 1 sent message to server
Msg sent by thread 1: Hello from thread 1
Time 0s: Thread 2 sent message to server
Time 0s: Thread 3 sent message to server
Msg sent by thread 3: Hello from thread 3
Msg sent by thread 2: Hello from thread 2
Time 0s: Thread 4 sent message to server
Time 0s: Thread 5 sent message to server
Msg sent by thread 4: Hello from thread 4
Msg sent by thread 5: Hello from thread 5
Time 4s: Thread 1 received message from server
Msg received by thread 1: Hello from thread 1
Time 6s: Thread 3 received message from server
Msg received by thread 3: Hello from thread 3
Time 11s: Thread 2 received message from server
Msg received by thread 2: Hello from thread 2
Time 12s: Thread 4 received message from server
Msg received by thread 4: Hello from thread 4
Time 14s: Thread 5 received message from server
Msg received by thread 5: Hello from thread 5
```