a1-LEP-25fa

Start Assignment

- Due Sep 21 by 11:59pm
- Points 15
- Submitting a website url
- Available Sep 3 at 9am - Sep 24 at 11:59pm

Any ambiguities must be reported and discussed on **the discussion thread (https://sjsu.instructure.com/courses/1613059/discussion_topics/5618689)** . (We want to experience the process of defining a "standard" implementation of a custom application protocol.)

What is The Leader Election Problem?

- In distributed computing, leader election is the process of designating a single process as the organizer of some task distributed among several computers (nodes).
- **Reference (https://en.wikipedia.org/wiki/Leader_election#)**

Definition

A valid leader election algorithm must meet the following conditions:

1. Termination: the algorithm should finish within a finite time once the leader is selected.
2. Uniqueness: there is exactly one process that considers itself as leader.
3. Agreement: all other processes know who the leader is.

Generating Unique ID

- A Universally Unique Identifier (UUID) is a 128-bit label used for identification in computer systems.
- When generated according to the standard methods, UUIDs are, for practical purposes, unique.
- **Reference (https://en.wikipedia.org/wiki/Universally_unique_identifier)**
- **Python Implementation (https://docs.python.org/3/library/uuid.html#uuid.uuid4)**

- We are going to use the `uuid.uuid4()` method to decide an ID of a process

Task 1:

Node Configuration

- We assume an asynchronous non-anonymous ring.
- Each node has exactly two neighbors.
  - Your code should include both client and server functionalities.
    - As a server, you will wait for another node to connect to your node.
    - As a client, you will connect to another node.
  - You will exchange a pair of IP address and port number outside of the code in class.
  - You prepare a simple text file `config.txt` in the same directory as the code. The configuration file should look like:

```
10.1.1.1,5001
10.1.1.2,5001
```

    - The first line should include your IP address (as a server).
    - The second line is the info exchanged with another student (as a client).
  - When you run the code, the configuration file should be used to initialize the connections. (You may want to set a reasonable length of sleep time to wait for a server node to be up.)

Connection

- Once you establish a socket connection, please maintain the connection between the neighbors for further communications. (Do not accept or ask for a new connection.)

Multithreading

- When you `accept` a connection, the server process needs to be run in a separate thread.
- This is because, if the single thread program has `accept` and `connect` in a sequence (as follows), there is no one to start the client connection, while waiting as a server with the `accept`

```
1: server.accept()
2: client.connect()
```

- After a connection is established, you can use single threading (or keep the multiple threads with a shared memory).

Algorithm

- Your node performs the O(n^2) algorithm.
- Read the first two paragraphs under **the *Asynchronous ring* section (https://en.wikipedia.org/wiki/Leader_election#Asynchronous_ring)** in the Wikipedia reference article.
- Sending direction
  - We decide the direction of the ring based on the client-server relationship. (This will be done outside of the code.)
  - You can assume that a message is always sent from a client to the corresponding server.
  - You should receive messages as a server and send messages as a client.
- Once you, as a client, are connected to a server, you should send a message with your uuid (without any comparison) as the initial message. This only happens once.

Message Format

- You must define a `Message` class, which has two member variables:
  - `uuid.UUID uuid`: indicating the sender's UUID. Note: This ID should be the same throughout the leader election process. (e.g. `123e4567-e89b-42d3-a456-556642440000`)
  - `int flag`: representing if the leader is already elected.
    - `0` if it is still in the process of leader election (initial value).
    - `1` if a leader is already elected.
- As discussed in class, we use **JSON to serialize the Message instances**. Do not change the aforementioned variable names.
- Each JSON message should end with "\n". At the receiving host, you need to keep reading the socket stream until it reads "\n".

Termination

- When enough time passes, every node in the ring (including your node) should have stopped sending messages (Termination condition) and have the same ID in a member variable named `leader_id` (Uniqueness and Agreement conditions).
- When terminating, the node should print something like `"leader is <leader_id>"`

Log

- When a process receives a message, it should clearly show, on a log file `log.txt`,
  - "Received"
  - `uuid` in the message
  - `flag`
  - if the message's uuid is greater than the process's uuid(print greater, same, or less).
  - if this process is in state 0 (still trying to find a leader) or state 1 (it knows the leader's ID).
    - If it is in state 1, show the leader's ID
- When a process ignores the message, it should clearly show, on a log file, that the received message was ignored.
- When a process sends a message, it should clearly show, on a log file,
  - "Sent"
  - `uuid` in the message
  - `flag`

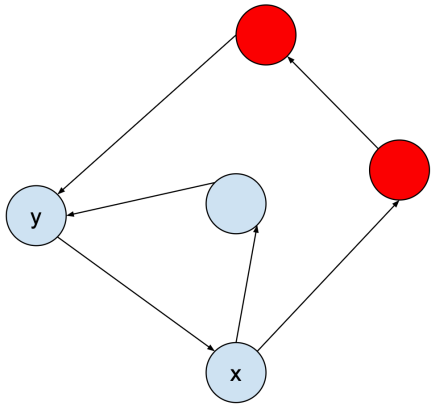
```
Received: uuid=f81d4fae-7dec-11d0-a765-00a0c91e6bf6, flag=0, less, 0
Received: uuid=f81d4fae-dddd-11d0-a765-00a0c91e6bf6, flag=0, greater, 0
Sent: uuid=f81d4fae-dddd-11d0-a765-00a0c91e6bf6, flag=0
...
```

```
...
Leader is decided to f81d4fae-dddd-11d0-a765-00a0c91e6bf6.
Sent: uuid=f81d4fae-dddd-11d0-a765-00a0c91e6bf6, flag=1
...
```

Task 1 - Demo

- Using three duplicates of your process implementation, create a ring and execute the election process.

Task 2:



Node Configuration

- **Now, you extend your program to handle a more complex topology, named a partially double ring. (See the image above.)**
- Before running the program, we randomly pick two nodes (x and y) to create the double ring.
- Your code should include both client and server functionalities.
  - As a server, you will wait for another node to connect to your node. If your node is "y", you need to wait for two connections.
  - As a client, you will connect to another node. If your node is "x", you need to connect to the second server.
  - You will exchange a pair of IP address(es) and port number outside of the code in class.
  - You prepare a simple text file `config.txt` in the same directory as the code. The configuration file should look like:

```
10.1.1.1,5001
10.1.1.2,5001
10.1.1.3,5001 (only if your node is "x".)
```

  - The first line should include your IP address (as a server).
  - The second line is the info exchanged with another student (as a client).
  - The third line is the info exchanged with another student (as a client), if you are connecting to two servers as node "x".
- When you run the code, the configuration file should be used to initialize the connections. (You may want to set a reasonable length of sleep time to wait for a server node to be up.) Also, when you run, you should be able to specify which type the node is, using the command-line argument as follows. [added on 9/5/2025]

```
python myleprocess.py y  # to run node y
python myleprocess.py x  # to run node x
```

```
python myleprocess.py n  # to run node n
```

Task 2 - Demo

- Using the five duplicates of your process implementation, create the five-node partially double ring (shown in the image) and execute the election process.

Submission Guide

- Use the GitHub classroom via this assignment link: **https://classroom.github.com/a/kd2D9bOv (https://classroom.github.com/a/kd2D9bOv)**
- You should have the following files in the GitHub repo:
  - myleprocess.py: the process to be part of the election ring
  - config.txt: the aforementioned config file
  - **task1** directory inclduing **log1.txt, log2.txt, log3.txt**: log files from the three processes in your local demo
  - **task2** directory inclduing **log1.txt, log2.txt, log3.txt, logx.txt, logy.txt**: log files from the five processes in your local demo (logx.txt should be from the node "x" and logx.txt should be from the "y".)
  - README.md: a brief explanation of how to run the programs. must include an execution example (copy, paste, and format the results shown on your terminals. It can be shown in the form of screenshots.)
- Source code should be appropriately commented. Also, we are going to check the modularity and readability of the code.
- Submit the link to your github repo to Canvas.

Evaluation in Class

- We are going to run the demos in class on Sept 22. Please be ready so you can run the process smoothly in class.