DNP Lab 06: Chord protocol

Important

You need to have at least three files:

- **main.py**. Implements the user interface.
- **node.py.** Stores Node class which implements the behavior of p2p node of Chord
- **registry.py**. Stores Registry class which Implements the behavior of Registry node.

Introduction

In this assignment you need to implement simplified version of Chord protocol.

1 Your main program

It should accept three arguments: *m*, *first_port,* and *last_port*.

- *m* represents the length of identifiers in chord (in bits) and has default value of 5.
- *first_port* and *last_port* represent the range of port numbers that the nodes are going to use. If first_port=5001 and last_port=5011, then 11 nodes should be created each of which listen at one of the 5001, 5002,…, 5011.

Your main program first should create the Registry and then nodes.

2 Registry

It is responsible to register and deregister the nodes. Its behavior is implemented by Registry class which is a child of Thread or Process class.

- It holds dict of id and port pairs of all registered messaging nodes.
- It has several methods:
  - **register** and **deregister** which are registered as RPC services at predefined port and can be invoked by remote messaging nodes.
  - **register(port)** method is registered as RPC service at predefined port and can be invoked by new node to register itself. It is responsible to register the new node with given port number, i.e., assigns id from identifier space. id is randomly chosen from [0, 2^m-1] range. Use random module and initialize it to seed=0 so your results will be the same as reference output! If newly generated id is the same as existing id, it again randomly selects another number until no collision
    - If successful, shall return tuple of (node's assigned id, optional message about the network size)
    - If unsuccessful, shall return tuple of (-1, error message explaining why it was unsuccessful)
      - It fails when for example when the Chord is full, i.e., there are already 2^m nodes given that m is the identifier length in bits
  - **deregister(id)** method is also registered as RPC service at predefined port and can be invoked by the registered node to deregister itself. It is responsible to deregister the node with given id. Returns tuple
    - (True, success message) upon successfully deregistering of the node with given id

    - (False, error message) upon failure. EX) no such id exists in dict of registered nodes
- **get_chord_info()** method is also registered as RPC service and usually is invoked by your main program to get the information about chord: dict of node id and port numbers.
- **populate_finger_table(id)** method is responsible to generate the list of the ids that the node with given id can communicate. It is registered as RPC service and can be accessed by the nodes.

3 Node

A usual p2p node of chord overlay. It is implemented by a Node class which is a child of either Thread or Process class. Node when created isn't isn't part of chord. It first registers itself using RPC invoking **register(port)** method of Registry class. Then after a second it populates its finger table invoking the populate_finger_table method of Registry through RPC.

It has at least following method:

- **get_finger_table()** which returns the finger table of the node which is just a list of node ids that it can communicate to. This method is registered as RPC service and accessible from the main program.
- **quit()** which is responsible to call the deregister(id) method of Registry to quit the chord and then shut down. quit() is also registered as RPC service and accessible from main program. Returns
    - (True, success message) upon successfully deregistering of the node with given id
    - (False, error message) upon failure. EX) no such id exists in dict of registered nodes

4 Example output

Important: Make sure that you initialized random module in registry.py with seed=0.

```
> python main.py 5 5000 5004

Registry and 5 nodes are created.

> get_chord_info

{26: 5000, 2:5001, 24:5002, 16: 5003, 31: 5004}

> get_finger_table 5000

[31, 8, 16]
```

Note: (*id: port*) combinations returned by **get_chord_info** can be different but make sure that only these ids: (26, 2, 24, 16, 31) are generated when you use seed=0.