M111 - Big Data - Spring 2022 - Programming Assignment

Alexandros Tavernarakis

This report will give a description on how the pub/sub architecture was implemented. As a starting point, the implementation was built on top of the sample Python code given in class.

Broker

The broker main() function launches two threads. One for communication with a subscriber and another one for communication with a publisher:

Thread1: subthread()

this function accepts connections from multiple subscribers as it uses the start_new_thread() function in order to create an extra new thread for each new subscriber that connects to the broker. Among other information, each subscriber provides to the broker the number of the port at which he listens for new messages. This way, the broker knows where to forward new messages from a publisher.

The new (sub)thread launches the function **threaded_client()** which first handles the command received by the subscriber (using the **commandline_breakdown_sub()** function) and creates a global dictionary, visible by both main threads, in which the topics at which every subscriber has registered, is saved.

Thread2: pubthread2()

this function is very similar to the subthread(). It accepts connections from multiple publishers. Similarly, a new (sub)thread is launched every time a new publisher connects. Each (sub)thread launches the function **threaded_client_p()** which has access to the global dictionary **mapping.** Every time a message is received by a publisher, firstly, it use the function **commandline_breakdown_pub()** which handles the publisher’s received command, and subsequently, it iterates through all the values of the dictionary and checks if a subscriber has subscribed to a topic. If the answer is yes, then it creates a temporary variable called **message4sub** and forwards it to the port at which the corresponding subscriber is listening to.

Furthermore, the function **create_socket()** is used by both threads to create a socket object, one for incoming connections from subscribers and another one for incoming connections from publishers.

By default, the broker (./broker.py) uses 9000, 9001, 127.0.0.1 for the parameters pub_port, sub_port, broker_IP, respectively.

Subscriber

The subscriber **main()** function launches two threads: one waiting for an input manually by the terminal (**user_input() function**) and another one, **subthread_sub()** which listens for incoming message (from the broker, at a specific port).

By default, the subscriber (./subscriber.py) uses s1, 8003, 127.0.0.1, 9001, “subscriber.cmd” for the parameters sub_ID, sub_port, broker_IP, broker_port, command_file, respectively.

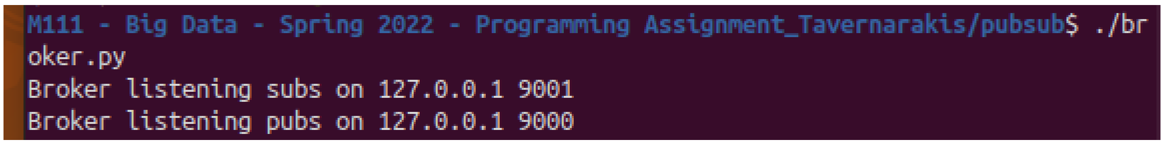
Publisher

The main difference of the publisher program is that there is no need for a multithread operation. It simply sends a command (either from a command file or from the terminal) and sends it to the broker via a specific port.

All three programs use the **argparse** python package in order to receive the user-defined parameters from the command line.

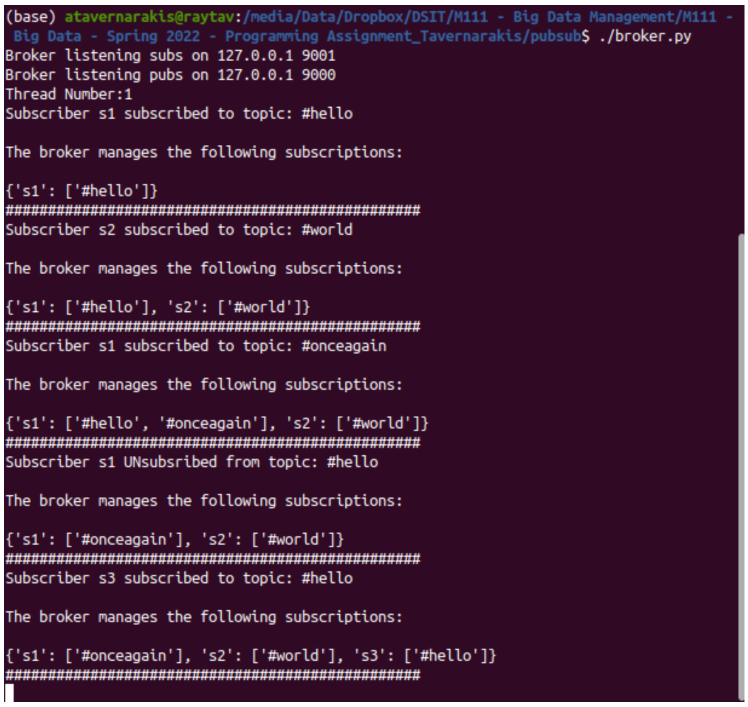
DEMO:

a. Launch the Broker: the broker sets up two different socket objects and listens for incoming connections at ports 9001 (for subscribers) and 9000 (for publishers)

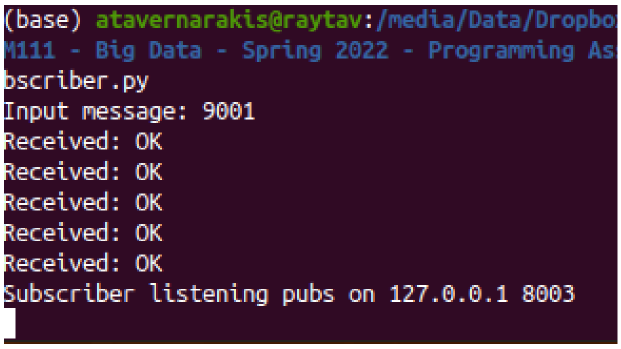


```
M111 - Big Data - Spring 2022 - Programming Assignment_Tavernarakis/pubsub$ ./br
oker.py
Broker listening subs on 127.0.0.1 9001
Broker listening pubs on 127.0.0.1 9000
```

b. Launch the Subscriber: the subscriber first load the subscriber.cmd command file and sends pre-defined commands to the broker. It therefore subsribes to specific topics as is indicated by the broker (left)



```
(base) atavernarakis@raytav:/media/Data/Dropbox/DSIT/M111 - Big Data Management/M111 -
 Big Data - Spring 2022 - Programming Assignment_Tavernarakis/pubsub$ ./broker.py
Broker listening subs on 127.0.0.1 9001
Broker listening pubs on 127.0.0.1 9000
Thread Number:1
Subscriber s1 subscribed to topic: #hello

The broker manages the following subscriptions:

{'s1': ['#hello']}
##################################################
Subscriber s2 subscribed to topic: #world

The broker manages the following subscriptions:

{'s1': ['#hello'], 's2': ['#world']}
##################################################
Subscriber s1 subscribed to topic: #onceagain

The broker manages the following subscriptions:

{'s1': ['#hello', '#onceagain'], 's2': ['#world']}
##################################################
Subscriber s1 UNsubsribed from topic: #hello

The broker manages the following subscriptions:

{'s1': ['#onceagain'], 's2': ['#world']}
##################################################
Subscriber s3 subscribed to topic: #hello

The broker manages the following subscriptions:

{'s1': ['#onceagain'], 's2': ['#world'], 's3': ['#hello']}
##################################################
```



```
(base) atavernarakis@raytav:/media/Data/Dropbo
M111 - Big Data - Spring 2022 - Programming As
bscriber.py
Input message: 9001
Received: OK
Received: OK
Received: OK
Received: OK
Received: OK
Subscriber listening pubs on 127.0.0.1 8003
```

It then stays on standby either for new user-defined commands by the terminal, or for incoming connections from the broker

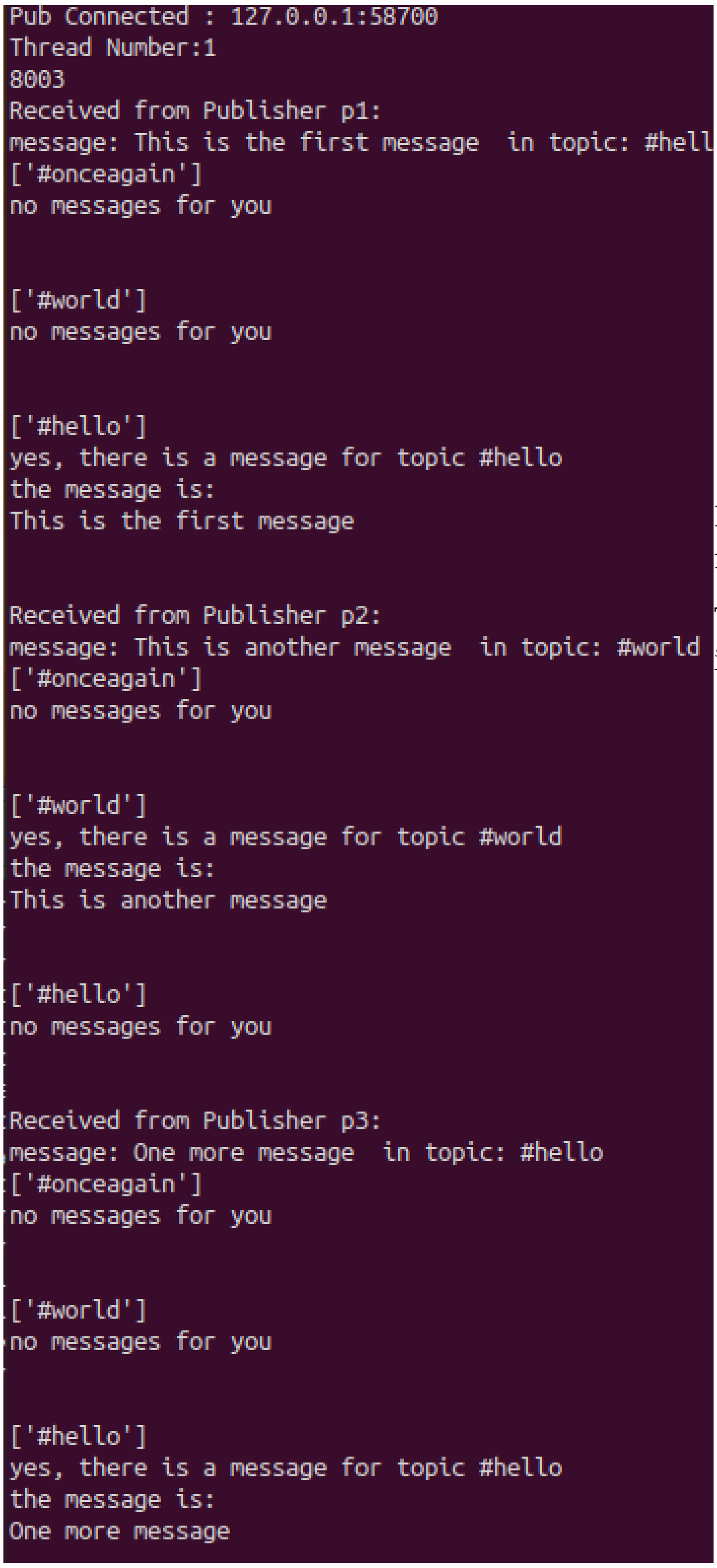
c. Launch the publisher

The publisher load the publisher.cmd command file and send the user-defined commands to the broker. The broker then, checks if there are any common topics with the subscribed ones and print the results on the screen. If there is a match it prints

“yes, there is a message for topic #xxxx”

“the message is:”

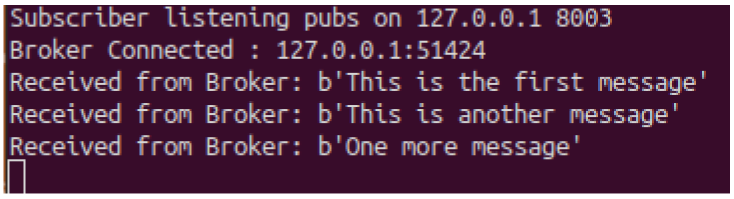
“……………………...”



```
Pub Connected : 127.0.0.1:58700
Thread Number:1
8003
Received from Publisher p1:
message: This is the first message  in topic: #hell
['#onceagain']
no messages for you


['#world']
no messages for you


['#hello']
yes, there is a message for topic #hello
the message is:
This is the first message


Received from Publisher p2:
message: This is another message  in topic: #world
['#onceagain']
no messages for you


['#world']
yes, there is a message for topic #world
the message is:
This is another message


['#hello']
no messages for you


Received from Publisher p3:
message: One more message  in topic: #hello
['#onceagain']
no messages for you


['#world']
no messages for you


['#hello']
yes, there is a message for topic #hello
the message is:
One more message
```

It then immediately connects to the subscriber and forwards the message.

The subscriber then receives the message and prints it on its screen:



```
Subscriber listening pubs on 127.0.0.1 8003
Broker Connected : 127.0.0.1:51424
Received from Broker: b'This is the first message'
Received from Broker: b'This is another message'
Received from Broker: b'One more message'
```

multicase:

The above pub/sub architecture permits the connection to the broker of multiple subscribers and publishers by creating a different thread for every connection which run in parallel and independently. Although, at each different thread there is interconnection of the different programs within the same thread, a feature that is missing is the communication between the threads which results in the broker being unable to forward the published message to all the different subscribers that have subscribed to a specific topic.