**Stressing my Kademlia implementation and diving into the code**

**Stressing the Kademlia implementation:**

First, we will have a look on how the Kademlia implementation manage stressful situations.

Because I don’t have a large IT infrastructure at my disposal, I will have to launch all the Kademlia instances on my laptop (11th Gen Intel Core i7-11390H @ 3.4 GHz, 16 GB RAM). Furthermore I will use Ubuntu on WSL 2 instead of Windows for obvious practical reasons.

Let’s write the following **stress.sh** bash script:

*#!/bin/bash*

*for i in {1..10000}*

*do*

*listening\_port=$(($i + 8081))*

*echo "I key$i value$i" | target/release/kademlia -l 127.0.0.1:$listening\_port -r 127.0.0.1:8080 2>&1 > /dev/null &*

*done*

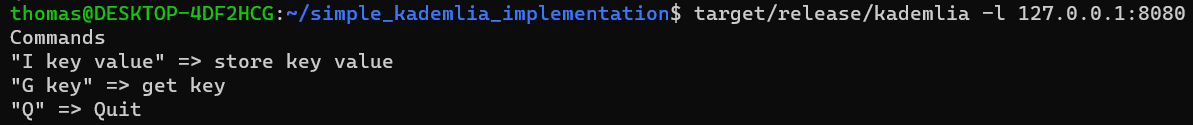
It will launch **10 000 instances** of Kademlia, with each an unique id **i**. Each instance will connect to the **entry point 127.0.0.1:8080** and listen on 127.0.0.1:{8081 + i}.

**Note:** On a real network, it’s preferable that each instance use a different entry point to join the network, but it doesn’t change anything here because the CPU will be used anyway.

Then the implementations will insert in the DHT **key{i} => value{i}**.

So let’s have a look on what happened.

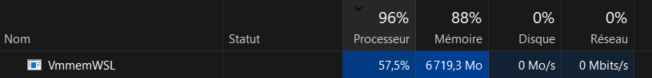
First of all, we **launch the first entry point** on a separated terminal:



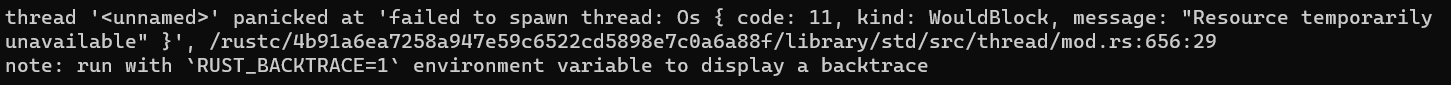
Then we launch the script:



Obviously I rapidly ran out of resources (even if the program is written in Rust):

.

Around last 80% of nodes just crashed because of the resources lack:



The 20% (2000) first nodes took a bunch of minutes to start. Because the Kademlia process runs in a separate thread, in means the **time bottleneck is the process and thread creation**, and not the connection to the Kademlia network.

**Let’s drive back to the first entry point**, and try to read values injected by the alive nodes.

If we try to read a value that hasn’t been injected into the DHT, the response “Not found” will take a while to appear:

Une image contenant texte, signe

Description générée automatiquement

However, if we try to read a key that has actually been injected, the response comes quasi instantly, no matter it’s number:





It means that finding an existing key is a lot quicker than looking for an nonexistent key, because the number of nodes to interrogate is a lot shorter.

Finally, let’s perform a lot of cleanup:

Une image contenant texte

Description générée automatiquement

**Conclusion:**

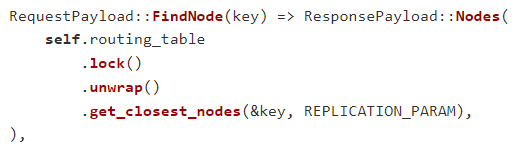
The performances test demonstrated that the Kademlia DHT network is really performant. Indeed, the performant bottleneck on a local machine was the thread and process creation, and node the Kademlia tasks themselves (joining the network and inserting / reading keys).

**Dive into the code:**

**Find node:**

Une image contenant texte

Description générée automatiquement



Une image contenant texte

Description générée automatiquement

So as we can see, the replication param is set to 20: *FindNode(key)* will return **a list of 20 nodes**, sorted by the XOR distance to the key. The buckets attribute is a growable list of “routing buckets”. So the algorithm will find the closest nodes in the nodes tree, with a logarithmic complexity. It’s a search-tree based on XOR distance.

**k-bucket structure:**

Une image contenant texte

Description générée automatiquement

As explained in the comment, the buckets are automatically refreshed.

**Reading and writing message:**

Une image contenant texte

Description générée automatiquement

It’s simply socket transmission over UDP, on the port specified in the command line argument.

**Ping:**

For pinging a node, the implementation just send a message with a special payload:

Une image contenant texte

Description générée automatiquement

And the other node responds as the same way:

Une image contenant texte

Description générée automatiquement

**Leave:**

There is no implementation of the leaving process, the node will just be removed after the refreshment period. However, there is a method to “kill” a node, meaning ask it to disconnect.