# Kyrys – performance analysis

To measure the bottlenecks of our application we used the tool *gcov*, which is part of the mingw distribution. This tool outputs the usage of each line of the source code to a file, which we can read and analyze.

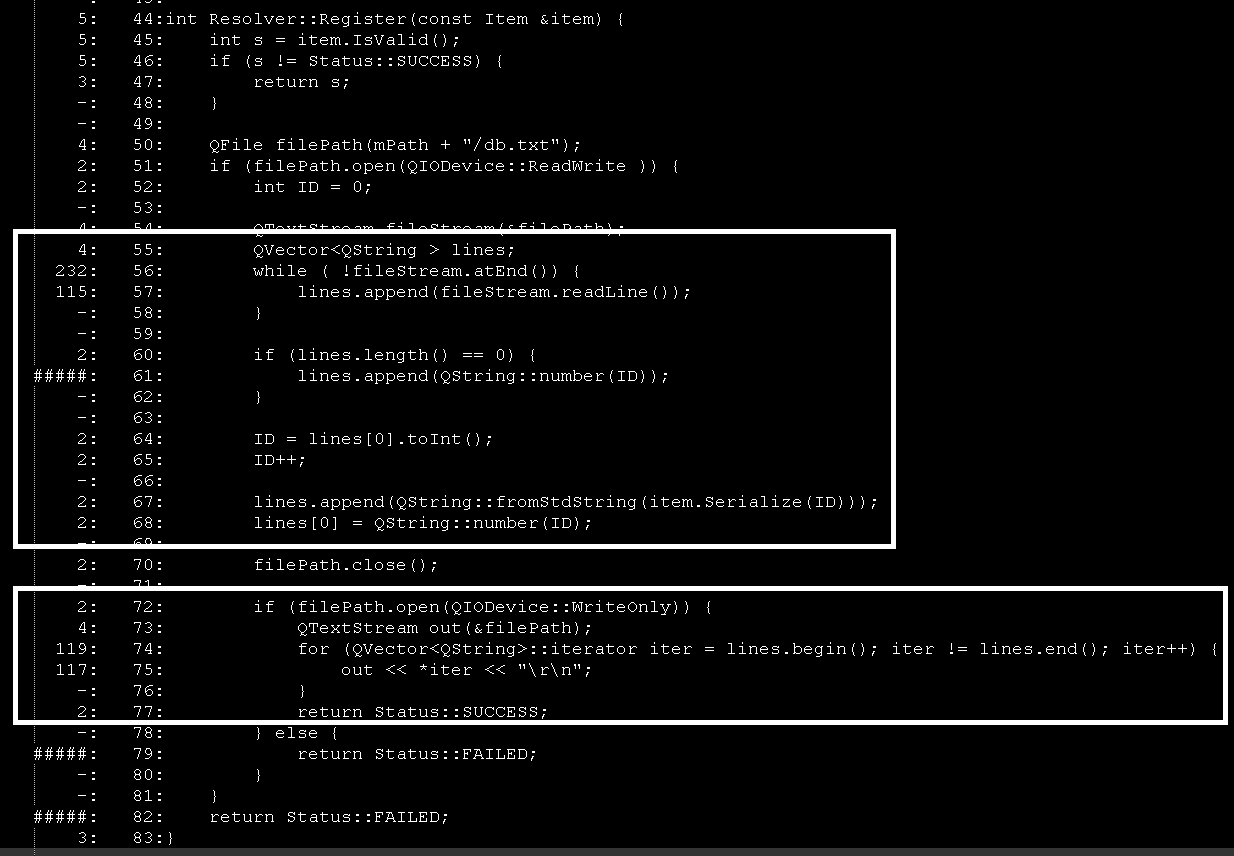
## Server – Resolver

This part of project parses and reacts to messages received by the server. The messages are received as JSON messages and parsed.

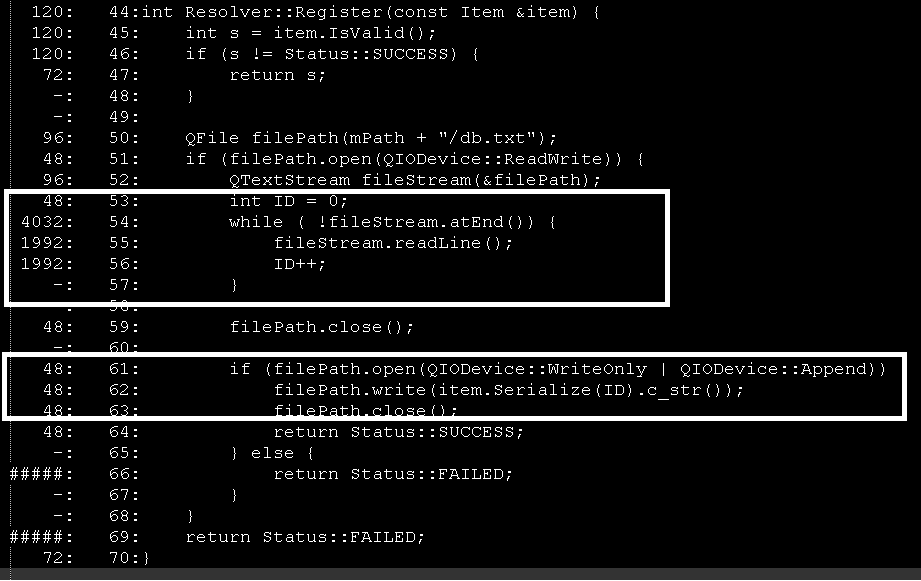
There are several functions in this class

* Parse() – this message tries to parse input string as JSON message, if it’s valid it executes particular command that is in the message. Only Qt functions are used in this part, no loops. Performance depends on the message size, Qt implementation and computer capabilities.
* Execute() – in Execute() the parser determines what method shall it use and uses that method. It’s implemented as a simple switch like structure, no loops.
* Register() – this function accepts an Item object as parameter and based on its content, it registers the user (aka adds user to database). This function reads the whole database (since it’s only a file) and appends the new user to its end.

In our former implementation there were two loops – one that fetched all the users from the file, then added one to the vector of users and the other loop traversed through the vector and dumped the whole vector back to file. This approach was rather ineffective, so we decided to ditch one of the loops and stick with only one loops, which counts the number of new lines in the file (aka number of registered users) and appends one new user to the end of the file. This optimization results in less running time (one less loop), less memory use (no vector needed) and higher readability of the code.

Before optimization, we can see usage of vector and two loops, one of which can be removed.

After optimization, one loop was removed and there is no use of vector.



## Server

This component receives messages from sockets and passes these messages to Resolver. Not very difficult implementation so far. It also sends messages back to client.

## Item

Item is responsible for handling items, which represent what is supposed to happen inside the server. The constructor of this class accepts a JSON document and extracts information from this document. The performance depends on Qt implementation.

* Serialize() – this function serializes the content of the item to string, so it is possible to write the content to file. All it does is concatenates a string.
* IsValid() – checks whether all parts of Item are valid (are not empty etc.)

## Utils – Random

This class contains functions for work with random numbers, strings etc. We will probably extend it in the future, so far it only has one function – getRandomString().

* getRandomString() – creates a random string of desired length. Uses a Qt qrand() generator and contains a for loop which counts to the desired length of the random string (default length is 10). It generates a (pseudo) random number in each iteration of the loop, gets a number from the available characters at the random index and appends the character to the resulting string.