Distributed Vector

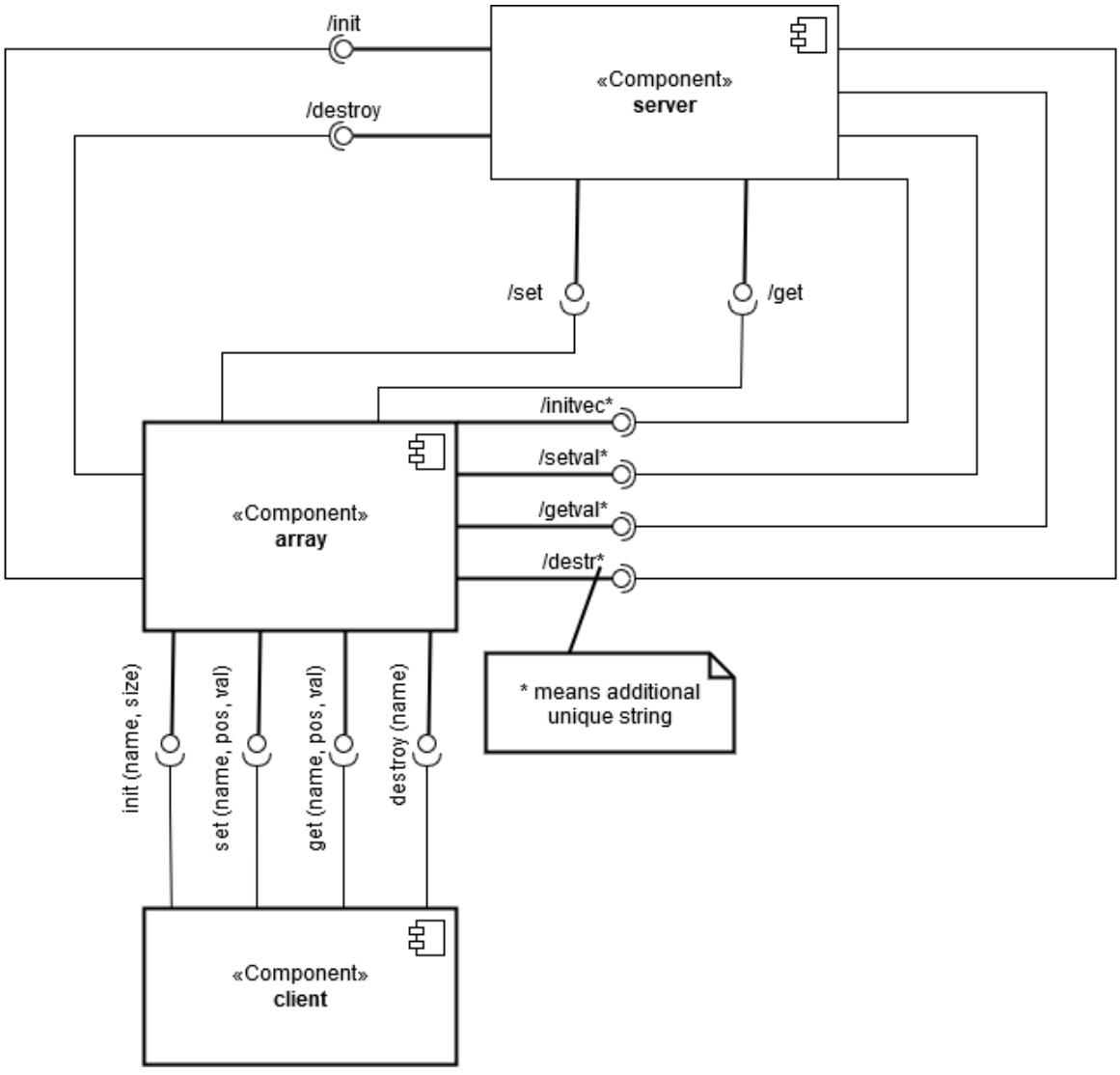
Author: Szymon Niemczyk

Design

The project consists of 3 main parts: server, client and array (interface between the server and clients). The following sections focus on each of them separately.

Server

The server creates 4 queues, one for every request: init, set, get and destroy. The queues are configured in a nonblocking way so that the main server thread can continiuously loop over them to check whether some requests appeared. This design solution makes it easy for even bigger parallelization – each queue could have separate thread (but it is just a possibility, it is not implemented like that).



Array

Array exposes 4 functions to clients: init, set, get and destroy. When array communicates with the server it creates response queues with different prefixes for each of the exposed functions. The prefix is followed by a number, which makes the queue unique in the entire system (in case there are multiple clients or client is using multiple threads to call the server).

Client

Client communicates with the server only through the array interface. First it runs the required sentence of calls and then performs additional tests to check whether server responds correctly, also when multiple threads are used.

Storage

Vectors are stored in vectors/ folder, each vector has a separate file. Access to each file is protected with a mutex. In order to get rid of mutexes which do not have corresponding files anymore there is a mechanism which counts how many threads use the mutex, and if mutex is marked to be deleted and the last thread unlocks it, then it is deleted.

Used libraries

The server takes advantage of a library providing vector (resizeable array) implementation. The library is available at https://github.com/Mashpoe/c-vector under BSD 3-Clause License.

Compilation and build

Server

```
gcc -pthread -o server server.c vec.o -lrt
```

Array

```
gcc -c array.c -o array.o -lrt

ar -rc libarray.a array.o
```

Client

```
gcc -pthread -o client client.c -L. -larray -lrt
```

Run and usage

1. Start the server ./server
   1.1. In order to stop the server write **q** and press **enter**
2. Start the client ./client