Micro Redis

You are to design and build a Redis-compatible service in Rust. Your service only needs to support a (small) subset of Redis commands. For pedagogical reasons, the core requirements below must be implemented even though the actual Redis server doesn't follow them faithfully (if at all).

Core Design Requirements

- **Single-machine service.** We will even make the assumption that the underlying hardware won't crash and the underlying software stack (e.g., OS, Filesystem, etc.) isn't going wrong, so the only source of issues (for now) will be our own code.
- **Multithreaded asynchronous.** The service will be asynchronous by design and take advantage of multithreading in places where that makes sense. But this should be automatic given that we're using `tokio`.
- **Linearizable.** The service guarantees the interaction (request/response) is linearizable under the failure mode assumption above. If it crashes, it will stop and give up (but it shouldn't crash).
- **Multi Databases.** The service supports keeping multiple *independent* databases (namespaces). Like real Redis, the `SELECT` command can be used to select which namespace to work on; there are only 16 databases available, numbered 0, 1, ..., 15. The default namespace is 0.

Wire Protocol. For our service to be Redis-compatible, it needs to follow the wire protocol specification, which is described at https://redis.io/topics/protocol. We'll use protocol version 2.

API Specifications

We'll refer to the official Redis command specification (https://redis.io/docs/latest/commands/) for the description of each command and its expected behaviors/performance guarantees. Properly handle errors that may arise. We will only support the following commands:

- **SELECT** - As described in the real spec. However, our service will only allow `SELECT` to be called at the start of the session (when a client first connects).
- **GET** - As described in the real spec.
- **SET** - As described in the real spec. However, we won't support any of the options, just the barebones `SET key value` command.
- **PING** - As described in the real spec. This command will be excluded from the linearizability requirement, i.e., won't be counted in the history.
- **EXISTS** - As described in the real spec.
- **RPUSH, LPUSH** - As described in the real spec.
- **BLPOP, BRPOP** (with timeout) - As described in the real spec. The number of keys that can be "checked" in one call will be limited to 5. Our timeout is interpreted as a floating-number (double).

As part of the development process, you must write extensive programatic tests to verify that your service meets the requirements. Automatic linearizability checking is optional (see the extra-work section below).

Exclusionary

You do *not* need to support pipelined operations. This is where for the same client, multiple commands are accepted at once—and while operation A is taking place, operation B is already lined up to be processed. For our service, each client will send a command and have to wait for the response to come back before the next command will be accepted by the service.

Getting Started & Hand-in Instructions

We'll use GitHub Classroom for projects.

- Use the join code from Canvas to make a repository.
- Following good commit/branching practices
- You'll tag what you want to submit with tag `1.0.0`

Tips and Pointers:

- Mini redis in Rust - https://tokio.rs/tokio/tutorial/setup