High level design

The client, *workerctl,* is a command line interface that talks to the worker server over HTTPS, meaning there is no persistent connection between them. After issuing a command, for example *workerctl start "ls -l",* the client sends an async POST request to the server.

On the server, after authenticating the user request, a goroutine gets created to handle the request, and, since commands may take a long time to execute, the server immediately sends a response to the client that contains the jobID and status "created.

To get status/output of the process, the client will send another request with the jobID returned from the previous command.

All job goroutines will have a timeout of 60 seconds to prevent them from running too long.

Server representation of a job:

```
type job struct {
    mu      sync.RWMutex
    id      string
    userid  string
    command string
    status  string
    output  []string
}
```

Security

Authentication will be handled by a pre-generated token saved in the repo. In the real world we would want to use an authentication service/package and a database to store tokens after login. Each API request will be authenticated first, before performing the request.

Authorization is the same for all registered users, they all can start/stop/check status of their own processes only.

All interaction between the client and the server are over **HTTPS**, where the client validates the server's credentials. The server doesn't care about the client's identity.

API

All API requests will be POST. The difference between requests will be in the body.

Request and Response Parameters:
action - ["start", "stop", "status"]

status - ["created", "running", "finished", "stopped", "error"]
token - authentication and authorization string
output - process output from stdout and stderr
error - API error message

Start a job
request: {"action": "start", "token": <token>, "command": <command>}
success response: {"jobID": <jobID>, "status": "created"}
error response: {"error": <errorMessage>}

Stop a job
request: {"action": "stop", "token": <token>, "jobID": <jobID>"}
success response: {"jobID": <jobID>, "status": "stopped"}
error response: {"error": <errorMessage>}

A job can only be stopped from the "created" or "running" states. if the job is in the "finished" or "error" states, it remains unchanged.

Job status
request: {"action": "status", "token": <token>, "jobID": <jobID>"}
success response: {"jobID": <jobID>, "status": <status>, "output": <output>}
error response: {"error": <errorMessage>}

Client

All requests sent be the client are asynchronous, allowing for submitting a batch of commands. It will be the responsibility of the user to track requests and responses.

Start a job
workerctl start "ls -l"
```
jobID: 7asd79as87d9asdas86d
status: created
```

Stop a job
workerctl stop <jobID>
```
jobID: 7asd79as87d9asdas86d
status: stopped
```

Job status
workerctl status <jobID>
```
jobID: 7asd79as87d9asdas86d
status: finished
output:
drwx------@  3 user  staff    96 28 Jan 20:59 Applications
drwx------@ 27 user  staff   864 14 Apr 11:57 Desktop
```

Scope and Trade-offs

To write as little code as possible, I will:

- hardcode a secret token for authentication
- write one or two tests for authentication and networking
- not isolate user environments on the server
- not create a configuration file
- not make it scalable or highly available

Future considerations:
For a real-world product, we would want each user to have their own isolated environment for running Linux commands. This could be achieved by using docker containers. After a user authenticates with the server for the first time, a docker container would be spun up and assigned to the user. After some time of inactivity, the container would be stopped, preserving the state for future requests. To prevent resource hogging we would limit resources assigned to each container.

For a more real-time experience, we would want the server to send updates to the client whenever there is new output from a process, and for that we could use gRPC.