Let's GO
Sophie Gabriela Laksø
GITHUB: https://github.com/tomat-suppe/disys-handin5/tree/main

**Introduction:**

This program is intended to simulate a leader crash, in which case a back-up server takes over. The server-change is invisible to the client, and only servers are able to see whether they have crashed or been activated.

The program is implemented with two servers (a primary server (serverside.go) and a backup server (backupserver.go)). All Bidder clients are within the clientside class, but act as different Bidders, just using same kind of basic code.

The server logs Bid and Result responses to a log placed at /tmp/logs.txt. To see this log after having run the program:

In the terminal, write "cat /tmp/logs.txt"

Otherwise, observe the logs in the clientside and the server terminals, or simply refer to attached pictures in this report.

**Architecture:**

I have implemented 2 servers as was the minimum requirement. The main server (serverside.go) is the original leader and is listening and serving as soon as the code is started.

The backup-server (backupserver.go) however is initialized as simply prompting the port whether a server is running. As such, as soon as there has been no server running on the port for 1 second, the backup-server turns on (starts listening and serving).

Clients can bid on the auction and query the result. I have implemented grpc, where the main message is the Bidder item, as Client always simply sends a Bidder, regardless of whether Bid() or Result() is called. Bidder has an id and a BidAmount, which is used for keeping track of which Bidder has bid how much, and to ensure that a Bidder will never bid a lower number than their own previous bids. They can however bid lower than *other* Bidders, in which case their bid is rejected!

A call to Result() will let the Bidder know whether the auction if over, and what the current/end highest bid was. Auction is over if 1 minute has passed.

My plan was to implement server failure after a set amount of time (in the code, I have set this to 15 seconds), however I can't really seem to figure out how to actually crash the server.

Though, if you start serverside.go, backupserver.go and clientside.go, then press ctrl+C in the serverside.go window, you can, in the command line log, see it shutting down and see the backupserver turn on in the backupserver command line log.

Let's GO
Sophie Gabriela Laksø
GITHUB: https://github.com/tomat-suppe/disys-handin5/tree/main

Servers and clients only communicate using grpc. I ran out of time on the last day unfortunately, but would have liked to re-make my proto-file, so I could send HighestBid data periodically to the backup server from the main server.

**Correctness 1:**

My implementation does not satisfy either linearizability or sequential consistency. This is because I didn't manage to implement the sending of data between servers before the hand-in time. In the case that this was implemented, I would send the data of HighestBid from main server to back-up server every time HighestBid was altered in the main server. This would then satisfy linearizability, as back-up server would immediately yield knowledge of HighestBid, and only start altering it itself when main server has failed.

**Correctness 2:**

Assuming no failure at all:

My main server runs entirely independently of the back-up server. My intention was to implement sending of HighestBid info from main server to back-up server, and even as such, program would function perfectly in an environment with no failures, though the back-up server would never end up being used.

In the presence of failure:

My protocol only tolerates one failure. As I have hardcoded two servers, and they can only do 'total failures', in the case of which both of the fails, there would be no protocol to start up a third server. It would probably be pretty easy to extend the protocol such that, upon failure of the back-up server, a new server would start and take over. I had begun implementing this at the start, with all servers being managed by the same class, but had to pivot in order to be at least almost done with the code in time.

In the case of only one failure (failure of the main server), the back-up server has almost the entirely same code as the main server and would therefore also be able to run without the main server. It is not strictly *correct* as I didn't manage to implement the sending of HighestBid from server to server, before the hand-in. Had this part been implemented, I reason that correctness holds in the case of only 1 failure.