CS Capstone Design

Technical Demo Grading Sheet (100 pts)

TEAM: 4 SSDynamics

Overview: The main purpose of the "Technical Demos" is to very clearly communicate the extent to which the team has identified key challenges in the project, and has proven solutions to those challenges. Grading is based on how complete/accurate the list of challenges is, and how convincingly and completely the given demos cover the given challenges.

This template is fleshed out by the team, approved by CS mentor, and brought to demo as a grading sheet.

Risky technical challenges

Based on our requirements acquisition work and current understanding of the problem and envisioned solution, the following are the key technical challenges that we will need to overcome in implementing our solution:

C1: Blocking Calls. Blocking calls create some unique challenges which include finding a way to realize when the command has finished, ensuring compatibility between different Opcodes, and deciding when a command should timeout and return an error.

C2: NVMe CLI interface. NVMe CLI interface is how NVMe calls can be called through python code. Its specific challenges occur with ensuring commands being called are called in the proper way to NVMe CLI through the use of admin passthrough. A proper demo of this would be to show a specific admin passthrough command running properly after being called in python code.

C3: Customized TLA+ interpreter in python. To consume the TLA+ specifications in Python, we will be using PlusPy. However, PlusPy's current logic does not solve what we need; programmatic control over simulation execution. As a result we will need some way to dissect PlusPy and create an interface utilizing PlusPy based off of it.

C4: Logging Output to File. All output is logged to a file. Everything from the interpreted TLA+ command, which goes to the Python program, and then runs the corresponding admin passthrough command through NVMe-CLI, timestamps and writes out to a file.

C5: Seeding and Resampling. While the Python program runs, it uses a seed to allow for running the exact same generated test again. There is a parameter that allows for a seed to be input which will run the exact same test. The seed creates a way to produce a unique test, and can be recreated later by using the same seed.

Challenges covered by demos:

In this section, we outline the demonstrations we have prepared, and exactly which of the challenge(s) each one of them proves a solution to.

Demonstration 1: Blocking calls

Challenges addressed: returning from blocking calls, ensuing compatibility between different Opcodes and when to timeout a command and return an error message

Flight Plan: Step by step overview of demo

1. First a simple command will be run, such as power on hours.
2. Then the program will halt until the power on hour counter is incremented
3. program will resume, “executing” the next command

Evaluation:

✔ Convincingly demo’d each of listed challenges?

✔ Other evaluative comments:

Demonstration 2: NVMe CLI interface

Challenges addressed: Ensuring that admin passthrough opcodes can be called through python.

Flight Plan:

1. First we will call an NVMe-cli admin pass through command call of a specific opcode
2. Then we will enter the same opcode information into the admin_passthru method in nvme_cli_interface.py and have a print to show the output
3. We will then run the file and show the output that occurs after the command is run
4. The two outputs will be compared and will show that the output of each is the same, so the python script is working properly to call the NVMe cli command

Evaluation:

✔ Convincingly demo’d each of listed challenges?

✔ Other evaluative comments:

Demonstration 3: TLA+ working custom interface

Challenges addressed: Custom TLA+ interface, programmatic control of model-simulator

Flight Plan: Step by step overview of demo

5. Run the custom interface, show available functionality (should be similar to plusPy, but more geared towards project)
6. Run step wise simulation
7. Run full simulation

Evaluation:

✔ Convincingly demo’d each of listed challenges?

✔ Other evaluative comments:

Other challenges recognized but not addressed by demo:

Logging Output to File and Seeding/Resampling(Challenges 4 & 5) are both challenges that haven't been fully recognized by our demo. While seeding and resampling can be demonstrated, we cannot guarantee that the output will match every time, and produce the exact output for the same seed. It will likely produce the correct output, but work hasn't exactly been done to verify that the seeding and resampling will work. We plan to mitigate this challenge with a lot of testing to make sure that the same output is produced before fully delivering.

Logging the Output to a File is something that hasn't been fully implemented, but something that will be implemented, since the rest of the output is currently more important than getting file output from the program. We are mitigating the risk of potential issues by making sure that the output matches exactly what we wrote to the terminal in our program, so that it can be reviewed for later, or studied if something went wrong with the program.