CO324 Lab 4: RPC APIs using gRPC

# **Objectives**

At the end of the lab you should be able to;

* Define services and messages using the Protocol Buffers interface definition language.
* Implement gRPC clients and servers.
* Proper input validation and error handling.
* Implement critical sections using mutual exclusion locks for concurrency control.

# **Instructions**

* Use only grpcio and built-in modules in Python 3 to complete these exercises.
* Solutions to discussion questions should be in a single plain text file named answers.txt. State your E-number at the top of answers.txt.
* Zip all files (including the ones provided) and submit them. No other archive formats please!

Submissions that do not follow instructions will incur a 10% penalty.

# **Preparation**

* Follow the instructions given in the [gRPC Python quickstart t](https://www.grpc.io/docs/languages/python/quickstart/)o set up your environment. Alternatively, you can use the following command (don’t mix the two sets of instructions!)

|  |
| --- |
| pipenv install grpcio grpcio-tools  pipenv shell |

* To check your setup download the helloworld.proto file and try compiling it. If everything is properly set up you should see two generated named helloworld\_pb2.py and helloworld\_pb2\_grpc.py

|  |
| --- |
| python3 -m grpc\_tools.protoc -I. --python\_out=. --grpc\_python\_out=. helloworld.proto |

* Now [run the “hello world” demo](https://github.com/grpc/grpc/tree/master/examples/python/helloworld) following the instructions in the python quickstart.
* (Optional) Install a [gRPC graphical client](https://github.com/uw-labs/bloomrpc), [VS-Code extension](https://github.com/oslabs-beta/tropicRPC), or the [command-line client.](https://github.com/fullstorydev/grpcurl)

# **References**

* Read the [introduction to gRPC](https://grpc.io/docs/what-is-grpc/introduction/) and [the core concepts.](https://grpc.io/docs/what-is-grpc/core-concepts/)
* [Protocol buffers v3 user guide](https://developers.google.com/protocol-buffers/docs/proto3) documents the interface description language.
* [Python gRPC generated code reference.](https://developers.google.com/protocol-buffers/docs/reference/python-generated)
* [Context Managers and the “with” Statement in Python](https://dbader.org/blog/python-context-managers-and-with-statement)

# **Exercises**

## **Part A: gRPC basics: deadline 8am, 26.10.2020**

1. Create a task list application that tracks tasks that a user needs to complete. Implement a gRPC API with the RPCs that are defined in the following proto file.

|  |
| --- |
| syntax = 'proto3';  /\* Task service API definition \*/  service Taskapi {  /\* Add a new task and return its id \*/  rpc addTask (TaskDesc) returns (Id);  /\* Delete a task by id \*/  rpc delTask(Id) returns (Task);  /\* List all tasks \*/  rpc listTasks (Empty) returns (Tasks);  }  message TaskDesc {  string description = 1;  }  message Task {  int64 id = 1;  string description = 2;  }  message Id {  int64 id = 1;  }  message Tasks {  repeated Task tasks = 1;  }  message Empty {  } |

You may use the skeleton code provided in the attached zip file to implement the server. The client that is provided will be used to test your submission.

## **Part B: Error handling, data races and lost updates: deadline 8am, 02.11.20**

Note: all exercises involve modifying task\_server.py only. task\_client.py is provided just for testing purposes.

1. State whether each Task API operation is idempotent along with an explanation.
2. Add error handling to the Task API RPCs to ensure the following.

* Task descriptions must be less than MAXLEN=1024 characters.
* Task IDs must be valid.

Return suitable status codes and messages if these conditions are violated.

1. Why is it necessary to implement the checks stated above?
2. Complete TaskapiImpl.\_\_enter\_\_ and TaskapiImpl.\_\_exit\_\_ to save and load tasks to and from a file.
3. Implement TaskapiImpl.editTask RPC that edits an existing task.
4. What happens if editTask is called on the same task by two clients simultaneously? Suggest a possible solution.
5. There is a subtle error in the provided implementation of Taskimpl.addTask called a [data race.](https://en.wikipedia.org/wiki/Race_condition#Example) How can we fix this problem?
6. A student writes the following code to prevent the data race in addTask. Explain why their code does NOT ensure mutual exclusion.

|  |
| --- |
| class TaskapiImpl:  def \_\_init\_\_(self, taskfile: str):  def addTask(self, request: StringValue, context) -> task\_pb2.Task:  with threading.Lock() as l:  t = task\_pb2.Task(id=self.task\_id, description=request.value)  self.tasks[self.task\_id] = t  self.task\_id += 1  return t |

1. Impose a critical section within your implementation of addTask to ensure proper mutual exclusion.
2. Does code in Taskimpl.listTasks need a critical section? Explain why (not?)