Faculty of Engineering and Architectural Science,

Department of Electrical, Computer and Biomedical Engineering

Lab Manual

COE892

Distributed and Cloud Computing

Dr. Muhammad Jaseemuddin

TAs: Amir Bagha, Parastoo Syed Jafarzadeh, and Abdul Bhutta

Winter 2024

Lab Schedule

Labs	Title	Tentative Due Weeks	Marks
Lab 1	Concurrency vs. Parallelism	Week 5	4
Lab 2	gRPC	Week 6	4
Lab 3	RabbitMQ	Week 8	4
Lab 4-5	FastAPI and Containers	Week 11	8

Pre-Lab

Software Environment

Following software environment is recommended and supported:

Ubuntu 22.04 LTS (https://ubuntu.com/download/desktop)
Python 3.0 or higher
PyCharm Community Edition (https://www.jetbrains.com/pycharm/)
gRPC (https://grpc.io/)
RabbitMQ (https://www.rabbitmq.com/)
FastAPI (https://fastapi.tiangolo.com/)

Lab Submission and Grading Policy

- You must submit lab individually
- Your submission includes a one-page note about your implementation illustrating main data structures and functions and screenshots of your test run
- A random sample of students will be called for demonstration for each lab
- Late submission will be accepted up to three days subject to the mark attrition rate of 10% per day of late submission

Lab 1: Concurrency vs Parallelism

Due Date: February 9, 2024

Objective

In this lab you will write two programs in Python using Python threading and multiprocessing modules to process data of several Land Mine Detector Robots.

Description

Consider the model of a land space as a 2D array of zeros and non-zeros where a positive value of (i, j)th element indicates that there is a mine at that location. You get the map.txt file where the first line has 2 numbers divided by space, the first one represents the number of rows of the land, and the second number indicates the number of columns. Then the rest of the file specifies the state of each cell in the land.

Part 1: Drawing the path of the Rovers

For this part, first you need to write a program which reads the map, then get the data of 10 rovers from the public API provided below:
https://coe892.reev.dev/lab1/rover/:number
Where the "**:number**" represents the rover's ID (ranging from 1 to 10). The response is a string containing all the commands that were sent to the rover. **List of the commands are:**

Commands	**Meaning**
L	Turn left
R	Turn right
M	Move forward
D	Dig

For example, the response could be **“RMLMMMMMDLMMRMD”**.

1. Write a sequential program that gets the commands of each robot and calculates the path of that robot (consider robots at point (0,0) facing south at the initial state). Create a file “**path_i.txt**” where **i** is the rover number and draw the path by using “*****” on the map, indicating the rover has been on that cell or 0 for the rest of the map. (You should consider if the rover hits a mine and it doesn’t dig it and move, the mine will explode, and the rest of the commands should be ignored). Using the time module, show how long it took to write all the path files.
   Note that if a Move Forward command is called when the rover is facing an edge, the command can be ignored.
2. Modify the program to take advantage of the Python Threading module to make the functions run in parallel. Record the total processing time and then compare it to the sequential approach and calculate the difference of their computation times. You should use Locks, if necessary.

Part 2: Digging mines

In this part, we want rover #1 to start digging and disarming any mine it comes across, upon receiving the digging (D) command.

To achieve this, write a program that reads the **mines.txt** file and considers the mine your rover is on in the file and their serial number. You can use a specific hash value to simulate the digging and disarming process.

To disarm a mine, you need to find a valid PIN by considering the following:

1. A selected **PIN** gets concatenated with the **serial number** of the mine, as a **prefix**, and results in a value we call **Temporary Mine Key**
2. The Temporary Mine Key is then hashed using the sha256 hash function, and a value is determined
3. If the determined hashed has at least **six leading zeros** (e.g. 00000A12B...), the PIN is **valid**

Please note that there may be more than one valid pin for the mines. To find a valid PIN by following the above procedure, you should use brute-force and start with any arbitrary string.

Your task is first to write a code for the above-mentioned process using a sequential approach in Python and record its time to disarm all the mines in the field. Then, write the same process but use Python

Threading module to run the functions in parallel. Record the total processing time and then compare it to the sequential approach and calculate their differences.

Lab Submission

You need to create a report file for the lab that would describe all the steps taken in each part. The report should include an introduction and conclusion section, as well. Please make sure all your files are compressed in a ZIP file (RAR is not accepted). The name of the file should be your group ID (e.g. “12.zip”). Only one D2L submission per group is accepted. All reports should be contained in the ZIP file and in PDF format (Docs is not accepted).

Lab 2: gRPC

Due Date: Feb 16, 2024

Objective

In this lab, you will write two programs in Python using gRPC to simulate the communication of several rovers with ground control, as an extension of Lab 1.

Description

Consider a gRPC server (the ground control) and 10 clients (the rovers) that are supposed to communicate using the gRPC protocol. Each rover implements the following methods:

1. Get map (e.g., “The 2D land array”)
2. Get a stream of commands (e.g., “RMLMMMMMDLMMRMD”)
3. Get a mine serial number
4. Let the server know if they have executed all commands successfully, or not (a mine might explode with a wrong series of commands)
5. Share a mine PIN with the server

For the sake of simplicity, note that each rover works independently; that is upon reaching a mine, a rover must disarm the mine irrespective to it being disarmed previously by another rover.

Part 1: Create the Proto Buffer Files

For this part, you need to create the *.proto* files required for the gRPC communication of the 5 methods mentioned in the description. You should describe the service, methods, messages and… required for the gRPC communication.

Part 2: Create the Server

In part 2, you need to create the gRPC server (the ground control) utilizes the 2D map files (as used in Lab 1), rovers command files containing the commands given to each rover (client) and a mine serial number file (similar to mines.txt seen in Lab 1). The server should be created based on the *.proto* files

generated in Part 1 of the lab and should implement at least the above 5 methods as mentioned in the description.

Part 3: Create the Client Side

In part 3, you need to create the gRPC client (the rovers) that should implement at least the above 5 methods mentioned in the description. This Python file should accept a CLI argument which indicates the rover number (1-10), which is expected when being executed. The rover (client) would retrieve the map upon running and store it in a suitable data structure. It would then retrieve the stream of commands and start executing them in order. This process does not need to use Python Threading and could be implemented sequentially. The client then needs to get the mine serial number, if it contacts a mine in need of disarming and share the mine PIN with the server, afterwards. Finally, the client needs to notify the server after executing all the commands successfully.

Lab Submission

You need to create a report file for the lab that would describe all the steps taken in each part. The report should include introduction and conclusion sections. Please make sure all your files are compressed in a ZIP file (RAR is not accepted). Submit the ZIP file to D2L. All reports should be included in the ZIP file and in the PDF format (Docs is not accepted).

Lab 3: RabbitMQ

Due Date: March 8, 2024

Objective

In this lab, you will write three programs in Python using gRPC and RabbitMQ to simulate the communication of a few rovers, the deminers and ground control together, as a continuation of Lab 2.

Description

In this lab, like the previous lab, the ground control relies on three files, the miners' serial number, the 2D map array, and the commands list. However, the rovers no longer perform the digging command. They only explore the land and request the gRPC server upon finding a mine to retrieve the mine serial number and publish a demining task using RabbitMQ. The entity of deminers, introduced in this lab, then use the queue to get coordinates of found mines, their ID, and their serial numbers. The deminers then start disarming the discovered mines, if not busy, one by one. Once a deminer had done, it would publish the PIN of the mine over another RabbitMQ channel, where the ground control is considered the consumer.

Part 1: Create the RabbitMQ server

In the first part, create the RabbitMQ server. You can create the server in different ways depending upon your environment. You can follow the instructions to download and install RabbitMQ server (and clients) given at Downloading and Installing RabbitMQ — RabbitMQ

Docker: Install docker on your machine, and then launch the server using the following command:

```
docker run --name mqserver -p 5672:5672 rabbitmq
```

This command creates a container from the **rabbitmq** image, named **mqserver**, and maps the port **5672** of the machine to the port **5672** of the container.

Part 2: Create the Ground Control

In this part, you need to create the server (the ground control) that takes advantage of the 2D map files (as used in Lab 1 and 2), rover command files containing the commands (excluding the digging command) given to each rover (client) and a mine serial number file (like mines.txt used in Lab 1 and 2). The server should be created based on the *.porto* files created in Part 1 of Lab 2, however we only require methods #1 to #3 (*Get map*, Get command stream, and Get mine serial number). The server should also subscribe to the **Defused-Mines** channel and either display the defused mine on a STD output file or log it to a file.

Part 3: Create the Rovers

In part 3, you need to create the gRPC client (the rovers) and implement the required methods mentioned in the description. This Python file should accept a CLI argument that indicates the rover number (1-10), which is expected when being executed. Upon running, the rover retrieves the map and stores it in a suitable data structure. It then retrieves the stream of commands and starts exploring the map. This process does not need to use Python Threading and could be implemented sequentially. The client then gets the mine's serial number, if it contacts a mine in need of disarming and publish the mine's coordinates, ID, and serial number to the **Demine-Queue** channel of RabbitMQ server.

Part 4: Create the Deminers

In this last part, you need to create a Python file for deminers. This Python file should accept a CLI argument that indicates the deminer number (1 or 2), which is expected when being executed. The deminers subscribe to the **Demine-Queue** channel of RabbitMQ server and then proceed to demine the assigned mine, if not busy. They then publish the PIN of the mine on the **Defused-Mines** channel of RabbitMQ server.

Lab Submission

Write a report for the lab that describes all the steps taken in each part. The report should include an introduction and conclusion section. Please make sure all your files are compressed in a ZIP file (RAR is not accepted). All reports should be contained in the ZIP file and in PDF format (Docs is not accepted).

Lab 4-5: FastAPI and Containers

Due Date: March 29, 2024

Objective

As a modification of Lab 2, in this lab, you will write two programs using FastAPI and WebSocket to simulate the RESTful communication of an operator with a server.

Description

In this lab, the server is a FastAPI HTTP server with endpoints available to use by the operator and rovers. The operator would be able to create, modify, delete, or dispatch a rover, or control it in real time, using the provided endpoints. In this lab, you should consider the rovers and the server as one module.

Upon receiving different requests, the server would simulate the rover movements, the demining action, and… so it could return the result back to the operator. All the communications of the server and the operator are in JSON format. **The endpoint list is as follows:**

Map		
Route	**Method**	**Description**
/map	GET	To retrieve the 2D array of the field.
/map	PUT	To update the height and width of the field.
Mines		
Route	**Method**	**Description**
/mines	GET	To retrieve the list of all mines, the response should include the serial number of the mines, and coordinates.
/mines/:id	GET	To retrieve a mine with the “:id”, the response should include the serial number of the mine, and coordinates.

/mines/:id	DELETE	To delete a mine with the ":id"
/mines	POST	To create a mine. The coordinates (X and Y), along with the serial number should be required in the body of the request. The ID of the mine should be returned in the response upon successful creation.
/mines/:id	PUT	To update a mine. The coordinates (X and Y), along with the serial number should be optional in the body of the request. Only the included parameters should get updated upon receiving the request. The response must include the full updated mine object.
Rover		
Route	**Method**	**Description**
/rovers	GET	To retrieve the list of all rovers, the response should at least include the ID and rover status ("Not Started", "Finished", "Moving", or "Eliminated").
/rovers/:id	GET	To retrieve a rover with the ":id", the response should include the ID, status ("Not Started", "Finished", "Moving", or "Eliminated"), latest position and list of commands of the rover.
/rovers	POST	To create a rover. The list of commands as a String should be required in the body of the request. The ID of the rover should be returned in the response upon successful creation.
/rovers/:id	DELETE	To delete a rover with the ":id"
/rovers/:id	PUT	To send the list of commands to a rover as a String. Note if the rover status is not "Not Started" nor "Finished", a failure response should be returned.
/rovers/:id/dispatch	POST	To dispatch a rover with the ":id", the response should include the ID, status, latest position and list of the executed commands of the rover.

Please note that ":parameter" in the routes, is considered as the parameter inside the path. Keep in mind that if any of the requests fail due to any reason, you must communicate the failure to the operator using HTTP codes, and a readable message in your own convention. Note that the map is clear of mines at the beginning, and it should get updated upon creation, update or deletion of any mine.

The implementation of the operator may be done using either Python or HTML and JavaScript. However, design of a user-friendly interface using HTML and JavaScript (executable on a browser) would qualify your submission for a bonus mark.

Part 1: Create the server

In the first part, create the server using Python and FastAPI. You need to implement all the endpoints listed in the lab description. Implement a suitable data structure to store all the required data (the map, list of rovers etc.). When using FastAPI, a route called “/docs” is created automatically which can be accessed for purposes of testing your implemented endpoints.

Part 2: Create the operator

In this part, you may create the operator in either Python or HTML and JavaScript. Designing a user-friendly interface (executable on a browser) would qualify your submission for a bonus mark. If you choose to use Python, you must still display the server responses in a readable and user-friendly manner. An example of a user-friendly web interface would be having a clickable map to select the coordinates of a mine, when creating, deleting, or updating a mine and updating the map upon success. When dispatching a rover, the client should display the rover path using the stars system used in Lab 1, or the mentioned map format on the web example.

Part 3: Implement the WebSocket [Optional]

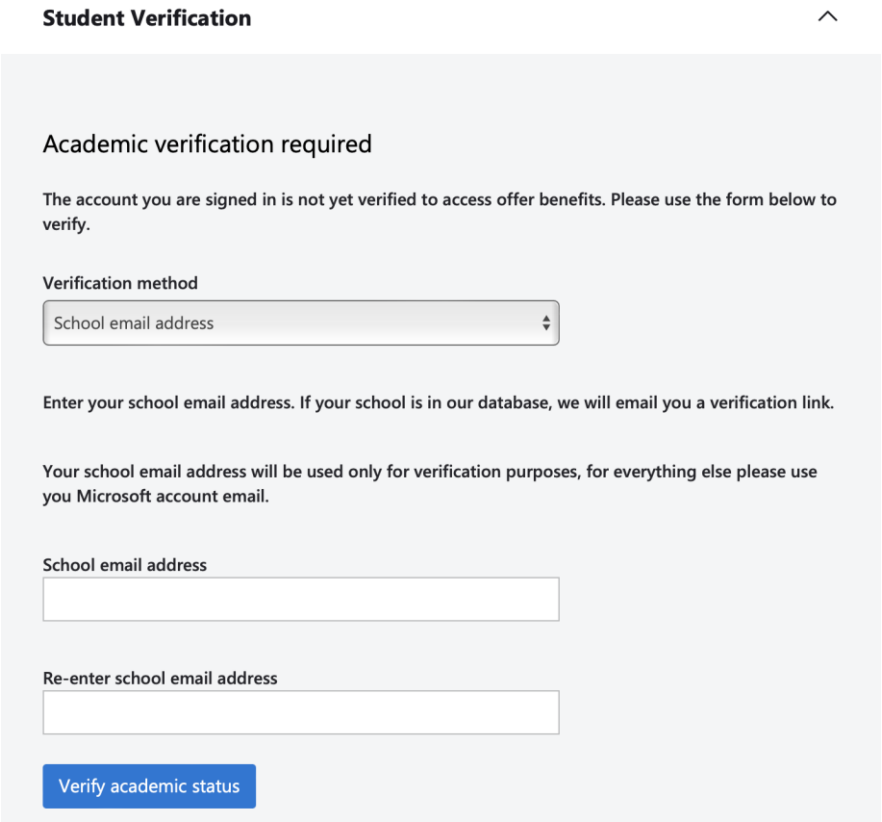
In part 3, you need to create a WebSocket endpoint in our FastAPI server. You can find an example of how to create a websocket endpoint in FastAPI at https://fastapi.tiangolo.com/advanced/websockets/ . This endpoint would enable us to control the rovers in real time. Upon receiving this request, the server needs to confirm that the rover is in a suitable status (“Not Started” or “Finished”) to execute the commands. In this state, the command list should be cleared for real-time control capability. After the connection is made successfully, the operator would be able to send the commands one by one to the rover (when L, R, M, or D keys are pressed). Upon receiving each command, the server must return a suitable response. For instance, for the L and R commands the server would return a true or false result, but for the M command, it may also return the new coordinates. By the same token, the server needs to return the PIN when the D command is called. The server must keep track of the rover status to manage the new requests, for instance it should not accept new commands when it’s executing a

previous one (the demining process may take time). In the meanwhile, the client should always update the interface based on the response of the server.

Note that the client may have multiple connections to multiple rovers at the same time.

Part 4: Azure Account Setup

In this part, you need to create an Azure account to deploy your program. First go to https://azure.microsoft.com/en-ca/free/students/ to create your account by clicking on “Start for free”. You’d then need to enter your school email address and click Verify academic status.



You would then receive an email **“[Microsoft Academic Verification] Confirming Your Academic Status”** which contains your verification link. You then need to create your profile. Now you should have access to the Azure portal dashboard.

4.1: Create a Docker Repository

Click on “Create a resource”, select “Containers” from the left menu, and select “Container Registry” to get started. Click “Create new” under the “Resource group” field and name it “Lab4”. Set the “Registry name” with the following format:

"COE892Lab42022G**ID**", where **ID** is your lab group ID.

Select the location as "Canada Central", and then click "Review + create", and then "Create". Wait for the deployment to be completed. Click "Go to resource" and then select "Access keys" from the left menu. Then, enable "Admin user". The information on this page would be used in the next steps to deploy your program.

4.2: Setup Docker Locally

Make sure you have Docker installed on your device and create a local container image from your server and make sure it functions as expected.

4.3: Deploy Your Container to the Remote Container Registry

Using docker login command against "Login server" found at the end of step 1. The command should look like the following:

```
docker login coe892lab42022g0.azurecr.io
```

Then, enter the username and password found at the end of step 1, and login should succeed. You now need to use the docker push command to push the container to the remote container registry. You then need to go back to the created COE892Lab42022GID resource page on Azure and select "Repositories" from the left menu. You can confirm that the upload has been successful.

4.4: Build a Web App using the Container Image

Click on "Create a resource" from the Azure dashboard, and search for "Web App for Containers" in the search bar of the page. Then, select the "Web App for Containers" option from the results and click "Create".

Now, select "Lab4" as the resource group and enter the web app name as "COE892Lab42022G**ID**", where **ID** is your lab group ID. Select "Canada Central" as the Region. Then, navigate to the "Docker" tab from the top and select "Azure Container Registry" as the Image Source as well as the deployed Image and Tag. Then click "Review + create", and then "Create". Wait for the deployment to be completed and click "Go to resource" and you should be able to use the URL to access your web application.

Lab Submission

You need to create a report file for the lab that would describe all the steps taken in each part. The report should include an introduction and conclusion section, as well. Please make sure all your files are compressed in a ZIP file (RAR is not accepted). The name of the file should be your group ID (e.g.,

“12.zip”). Only one D2L submission per group is accepted. All reports should be contained in the ZIP file and in PDF format (Docs is not accepted).

Lab Marking Scheme

Lab 4 [5 marks]: Part 1 and Part 2
Lab 5 [5 marks]: Part 4
Bonus [5 marks]: Part 3 – You can get the bonus and utilize it to increase your marks in the lab component of the course. Your overall lab component after adding bonus should not exceed 25% of the course evaluation. In other words, the bonus will help if you have shortfall in the lab component of the course evaluation.