Programming Assignment 1: Setting up your working environment.

1 Lab Task

In this lab, you will learn how to setup your working environment for the projects in this class and build a simple project that will serve as the basis for the next projects. This class requires you to use containers. You will go through a containers tutorial, then you will create your own scripts that you will use for the projects in the class.

1.1 PART 1: Containers tutorial

You are required to read and go through the exercises at:

`https://github.com/iowaguy/docker-tutorial`

1.2 PART 2: Write a base program

You need to implement this program in **C/C++**. The main goal of the project is for n programs to coordinate with each other via UDP messages before starting a task.

Your program will open a UDP socket, and wait for messages from the $n-1$ other programs announcing they have started and they are ready. The program will print "READY" **to stderr** (not stdout) when it has heard a message from each of the other $n-1$ programs. The program will read from a configuration file who the other processes are that they will need to receive messages from.

You will be using Docker and Docker Compose to package your program. Instructions on how to use Docker and Docker Compose for this project can be found in the Docker tutorial. You will be provided with a `docker-compose.yml` file. Your program should be able to use that file without modification.

Building and running instructions below. Note that you will need to build the Docker container before orchestrating with Docker Compose:

```
Building:
    docker build . -t prj1

    Running this in your project's directory should
    build your project's Docker image using your
    Dockerfile. It should copy over the code and
    hostfile to the image and compile your project.

Single container usage:
    docker run --name <hostname> --network <network> --hostname <hostname> prj1 -h <hostfile>


  --name <hostname>
    This specifies the name of the Docker container,
    this name should match the one in your hostfile
```

```
  --network <network>
    This is the user-defined Docker network your project
    will run in, refer to tutorial in additional instructions
    to learn how to create one and why you need one.

  --hostname <hostname>
    This specifies the hostname of the Docker container,
    this name should match the one in your hostfile. It can be used by other
    containers in the same bridge network (e.g., like the one created in the
    tutorial, or the one used in the Docker Compose template).

  -h <hostfile>
    The hostfile is the path to a file inside your Docker container
    that contains the list of hostnames that the Docker container is
    running as (specified using the --name flag above). It assumes
    that each container is running only one instance of the program.
    It should be in the following format.

    container1
    container2
    ...

    All the processes will listen on the same port.
    The line number indicates the identifier of the process
    which starts at 1.

Orchestration:
   docker compose -f [PATH TO COMPOSE FILE] up
```

You can choose your own design for implementation. For example, note that if a process starts sending messages before the others are up then they will miss the message. To avoid this situation, you can have processes sleep at the beginning, or send "I am alive" message multiple times and wait for an acknowledgment from the recipient. You can use events or multi-threading. Additionally, you can experiment with how you want to debug, log information, etc.

IMPORTANT: The functionality in this project and learning how to work with UDP sockets and Docker containers will be needed in your next projects, so it is important that your project will work. Submitting code that does not run, or does not run correctly will prevent you from doing the other projects.

IMPORTANT: All messages must be printed to `stderr` and all printed messages must be terminated with a newline (`\n`).

2 Grading

The project is manually graded by your TA and not by autograder. Total: 100 points.

- Writing the Dockerfile, starting your n processes with Docker Compose (without the networking part implemented): 30 points.
- Implementing the networking code: 60 points
- Code inspection: 10 points

3 Submission Instructions

Your submission must include the following files:

1. The **SOURCE** and **HEADER** files (no object files or binary)
2. A **Dockerfile** to build your project
3. A **hostsfile.txt** that we can use to test your project
4. A **README** file containing your name, any additional instructions to run your code and anything you would like us to know about your program (like errors, special conditions, etc.)
5. A **REPORT** describing your algorithm.

Submission is through Gradescope.

4 Recommendations

4.1 Network Programming

Most people will find communicating with UDP sockets the most difficult part of this project. UDP (User Datagram Protocol) is a connectionless networking protocol. If the IP protocol delivers packets from one computer to another, UDP delivers packets between processes running on different computers. Each party in transport protocols (like UDP and TCP) are identified by an IP address and a port. Connectionless means that the sender does not need to establish a connection with the receiver before sending message to them. UDP is an "unreliable" protocol, it does not guarantee that the packets will be delievered. If a packet is altered in transit, or dropped for any reason, UDP will make no attempt resend. Its advantage over a reliable protocol like TCP is that it is lightweight and allows implementation of customized protocols.

Network protocols like UDP and TCP are typically implemented in the operating system, and you can work with them through a set of system calls. You will need to use the following functions `getaddrinfo()`, `socket()`, `bind()`, `sendto()`, and `recvfrom()`. For details on how to use these functions, see the additional resources section, particularly, the socket programming link.

It is possible to do this assignment by providing IP addresses in the hostsfile, however, you will likely find it easier to use human-readable hostnames due to some complexities in Docker's virtual networking stack.

4.2 C/C++ Programming

C/C++ can be difficult languages because everything needs to be explicitly defined. Unlike high-level languages like Python, you need to tell the program exactly *how* it should interact with the operating system. You may find this challenging, but it is an excellent learning opportunity.

If you need to allocate memory dynamically (i.e., you won't know how much memory you need until runtime), you will need to do so explicitly. In C, you can use the functions `malloc()` or `calloc()`. Likewise, C++ has the keyword `new`. You may also find it useful to zero out the newly allocated memory blocks (use `memset()` in C) as freshly allocated blocks can contain unknown values from previous computations. It is also good practice to `free()` any allocated memory blocks when you are done with them (called `delete` in C++). Failure to do so will result in *memory leaks* which can cause your program to crash. For compiling your code, you will most likely want to use `gcc`.

For writing to stderr, you will file the `fprintf()` function useful. Remember, in Linux everything is a file, including stdin, stdout, and stderr.

4.3 Docker

The things you will want to do in your Dockerfile are the following

1. Start with a reasonable base image (ubuntu:22.04 is a good choice).
2. Install necessary packages.
3. Copy source files and hostsfile.txt over to container.
4. Compile code.
5. Set "ENTRYPOINT" to the path of your binary.

When running on MAC you might have issues with DNS lookups failing in Docker images. One of the symptoms can be that getaddrinfo function will fail. To fix DNS lookups in Docker create the file below with the following contents to set two DNS, firstly your network's DNS server, and secondly the Google DNS server to fall back to in case that server is not available:

`/etc/docker/daemon.json:`

```
{
    "dns": ["10.0.0.2", "8.8.8.8"]
}
```

Then restart the docker service:

```
sudo service docker restart
```

4.4 Development Advice

- Start with small pieces and test them frequently. Make sure one component is working as expected before moving on to the next one.
- Take advantage of functions to separate segments of code that are logically different. This will make things much easier for you to read and debug.
- There is an excellent C/C++ debugger called `gdb`. You can step through individual lines of code at runtime, print values, etc. Just make sure to compile your code in debug mode (`gcc` has a flag for this).
- Print all the info you are sending exactly before sending it over the network, then print it again when you receive it. Create functions that allow you to "pack", "unpack" information you are sending over the network and always use those functions to avoid mistakes.

5 Additional resources

You may find the following resources helpful

- Socket programming: `https://beej.us/guide/bgnet/html/`
- Unix programming links: `https://cse.buffalo.edu/~milun/unix.programming.html`
- C/C++ programming link: `https://www.cplusplus.com/`
- `https://github.com/iowaguy/docker-tutorial`