Virtual Development Environment in a Box

Students: Dylan McDougall (dmcdougall2019@my.fit.edu)

and Ian Orzel (iorzel2019@my.fit.edu)

Faculty: Ryan Stansifer (ryan@fit.edu)

Client: Ryan Stansifer (ryan@fit.edu)

Progress of current Milestone:

Task	Completion	Ian	Dylan	Todo
Basic Qemu Image for Compiler Theory	100%	1%	99%	Experiment with QEMU smb
Run Commands and Provide/Receive Standard Output/Input	100%	75%	25%	
Import and Export Files from an Image	100%	99%	1%	

Discussion:

- **Basic Qemu Image for Compiler Theory**:

  The compiler theory container requires access to the GNU Toolchain (gcc, make, etc.), GNU Binutils (as, ld, etc.), Java, and the C standard library. We wanted to include all of the required software within the container so that a prospective Compiler Theory student can all of their required software with minimal fuss. The GNU tools must be able to compile SPARC binaries, and the container must be capable of running those binaries. There are two approaches I tried to take when fulfilling these requirements

  Approach 1) Have the entire container exist within an emulated SPARC environment with all the required development tools included.

Approach 2) Provide Java and a cross-compiler, cross-assembler, cross-linker, etc. inside a container, and provide a SPARC emulator with the container.

The first approach seemed to be the simpler one, however OpenJDK is not officially supported on the SPARC architecture, and although compiling the jdk for SPARC may be theoretically possible, it is not at all practical.

The second approach is the one that was ultimately successful. I was able to build a cross-compiler for SPARC with support for the C standard library provided by **uClibc**. The containerized environment uses Linux's binfmt to allow SPARC programs to execute In the same manner as native programs. This container functions very similarly to andrew.cs.fit.edu.

I also experimented with providing support for MIPS emulation in the container as per a request by Dr. Stansifer, and was able to successfully get a working MIPS (little-endian) cross-compiler, along with the other cross-tools.

On non-Linux platforms, the use of a Linux virtual machine is necessary since I have only been able to reliably (or rather, at all) build cross-compilers for Linux. On Linux platforms or WSL, it is still advantageous since the virtual machine requires no setup and is entirely self-contained.

- **Run Commands and Provide/Receive Standard Output/Input**: Before embarking on this task, we had to create functionality to start and stop containers, as well as set up an SSH connection with these containers. Once we had this functionality covered, we used the SSH connection in order to send commands to the container. Receiving standard

output from the container has been working fine, but we have run into some issues with sending standard input. Currently, we have problems with sending signals to the process that standard input is completed as well as preventing standard input from being mirrored (while the standard output is not being buffered).

- **Import and Export Files from an Image**: For this task, we need to be able to send and receive files from a given container. To accomplish this, we utilized the SSH connection that we set up during the previous task. Using the Connection object in SSH's Fabric library, we are able to send and receive files using a basic command. We ran some tests, and we have ensured that these functions work as expected.

Member Discussion:

- **Ian Orzel**: During this milestone, I spent all of my time focusing on the container manager repository and creating the code base for this. Starting from the ground, I utilized Python to create a program that could manage containers. For this, I implemented a container class that had the ability to start/stop containers, send commands to containers while dealing with the input and output of them, and could send/receive files to/from a container. Then, I implemented a container manager class that managed different containers so that multiple containers could be supported at a time.
- **Dylan McDougall**: I spent most of my time after Milestone 1 working on the Compiler Theory container and I feel I have achieved the most practical solution for shipping the required tools across all relevant desktop platforms. Once the cross-compilers and Java were resolved, getting things like ssh and make were trivial. Once the command-line tool is fully implemented the building of the workflow will essentially be completed.

Regarding the running commands via ssh task, this has been giving us a little bit of trouble. Ideally, we would like to use a library to act as a middleman between ssh and our program, but this is proving to be difficult as all the mainstream ssh libraries are not designed for interactivity and more for automation. This is understandable but it is inconvenient for us. We are still figuring out how we're going to solve this. I have implemented a proof-of-concept solution that uses the host's OS shell to call ssh, but this is rather crude. Even so, we may not have another choice if we cannot solve the standard input issue, which it's looking like we won't be able to.

Next Milestone Matrix:

Task	Ian	Dylan
Implement, demo, and test command-line interface support in Python	99%	1%
Implement, demo, and test command-line interface as shell and batch scripts to send information to Python	50%	50%
Implement, demo, and test installer or installation guide for installing system on Windows, MacOS, and Debian	1%	99%

Discussion of Planned Tasks for Next Milestone:

- **Implement, demo, and test command-line interface support in Python**: Currently, the container manager is written as more of an API to be used by a programmer. Ideally, the user will be sending the Python program strings through the CLI, so the Python program needs to be able to parse these commands and convert them into the appropriate Python code to accomplish what the user wants.

- **Implement, demo, and test command-line interface as shell and batch scripts to send information to Python**: We do not want the user to be directly sending information to the Python program. We instead want the user to run a basic command that sends what they input to the Python program. To do this, we plan to write shell and batch scripts to act as a basic interface between the user and the Python program.
- **Implement, demo, and test installer or installation guide for installing system on Windows, MacOS, and Debian**: In order for users to use the tool, they must be able to install it on their local machine. Thus, we want to implement a way for users to easily install the system on their machine. The first (and ideal) option for this is to create an installer for each operating system, which is a program that will automatically install the system for the user. If that is not currently possible, we will create a basic installation guide, which will take the user through all the steps that they need to go through in order to install the system.

Dates of Meeting with Client: (See Faculty Meeting Times)

Client Feedback for Milestone: (See Faculty Feedback)

Dates of Meeting with Faculty:

- October 12
- October 19
- October 26

FacultyFeedback for Milestone:

- **Basic Qemu Image for Compiler Theory**:

- **Run Commands and Provide/Receive Standard Output/Input**:

- **Import and Export Files from an Image**:

Faculty Advisor Signature: _____ Date: _____

Score for each member:

Ian	0	0.5	1	1.5	2	2.5	3	3.5	4	4.5	5	5.5	6	6.5	7	7.5	8	8.5	9	9.5	10
Dylan	0	0.5	1	1.5	2	2.5	3	3.5	4	4.5	5	5.5	6	6.5	7	7.5	8	8.5	9	9.5	10

Faculty Advisor Signature: _____ Date: _____