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)

Dates of meetings:

- January 11 from 3:00 - 4:00 PM

Goal:

Our goal is to create a platform that allows students to run software inside of a virtual environment that can be configured to have system specifications different from the client system. This system should be easy to use for both students and faculty, should be scalable, and should support multiple platforms. Specifically, the system should be able to support the Compiler Theory class, allowing users to run Sparc binaries.

Motivation:

Our motivation is to minimize the difficulties related to setting up the environments for students to use during their coursework, especially when that software requires exotic hardware. This creates problems for students and faculty, as students have to spend time attempting to get the system working and faculty have to spend time helping them with this process. We also hope to minimize issues where students complete tasks on their system that their professors are not able to replicate.

Approach:

- Use on-demand virtual development environments custom-tailored to your courses. These virtual environments contain all the necessary custom software you need to

develop programs for assignments, as well as to test and debug your software, all wrapped up in a tidy container that you install once, and you're done.

- Use the pre-made environments created for the Compiler Theory, Programming Languages, and Operating Systems courses.
- Easily emulate foreign guest architectures other than that of your host machine without even having to think about it.
- Run these development environments on all the major platforms: Windows, macOS, and Linux.
- Transfer files between your host's native filesystem and the container's virtual filesystem in an intuitive manner.
- Access and interact with the virtual shell of the container just as you would your own.
- Be able to create containers with whatever custom software you need to distribute to your students for a specific course.

Algorithm and tools:

- We have ultimately decided to use QEMU for virtualization. We had a strong sense that we were going to use this software from the beginning because it had a number of distinct advantages. For one, it is completely open source, unlike competing software, which is either closed or only partially open source. It is also incredibly flexible. Virtual machines can easily be configured to run headlessly and are easy to distribute. QEMU is chiefly emulation software, which is advantageous to us since it allows us to not worry about architectural discrepancies.

- Our program is written in Python, and in order to distribute a usable version of this Python code, we are using something called PyInstaller. PyInstaller allows us to package up our Python code, along with all of its dependencies, into native self-extracting executable files. Using this tool, the end-user need not even be concerned that the program is written in Python, since it packages up the Python project in a way that allows it to be interacted with just like any natively compiled program.
- The project is split into two distinct but heavily interdependent components, which communicate via standard UNIX TCP sockets, in the form of a kind of client-server relationship, but both the client and server are run on localhost and never communicate with the outside internet (except in the case of repositories.) Our reasoning for this is simple. One component of the program needs to always run in the background, in the form of a daemon, in order to manage currently open containers. The other component must be executed on-demand by the user via the command line, and be able to send requests to the other component. Sockets are the most convenient way for these two components to communicate on all supported platforms.
- We are using SSH as a means to access the shell of the virtual machine. This is fairly standard and easy to implement since SSH is pretty much universal.

Novel features:

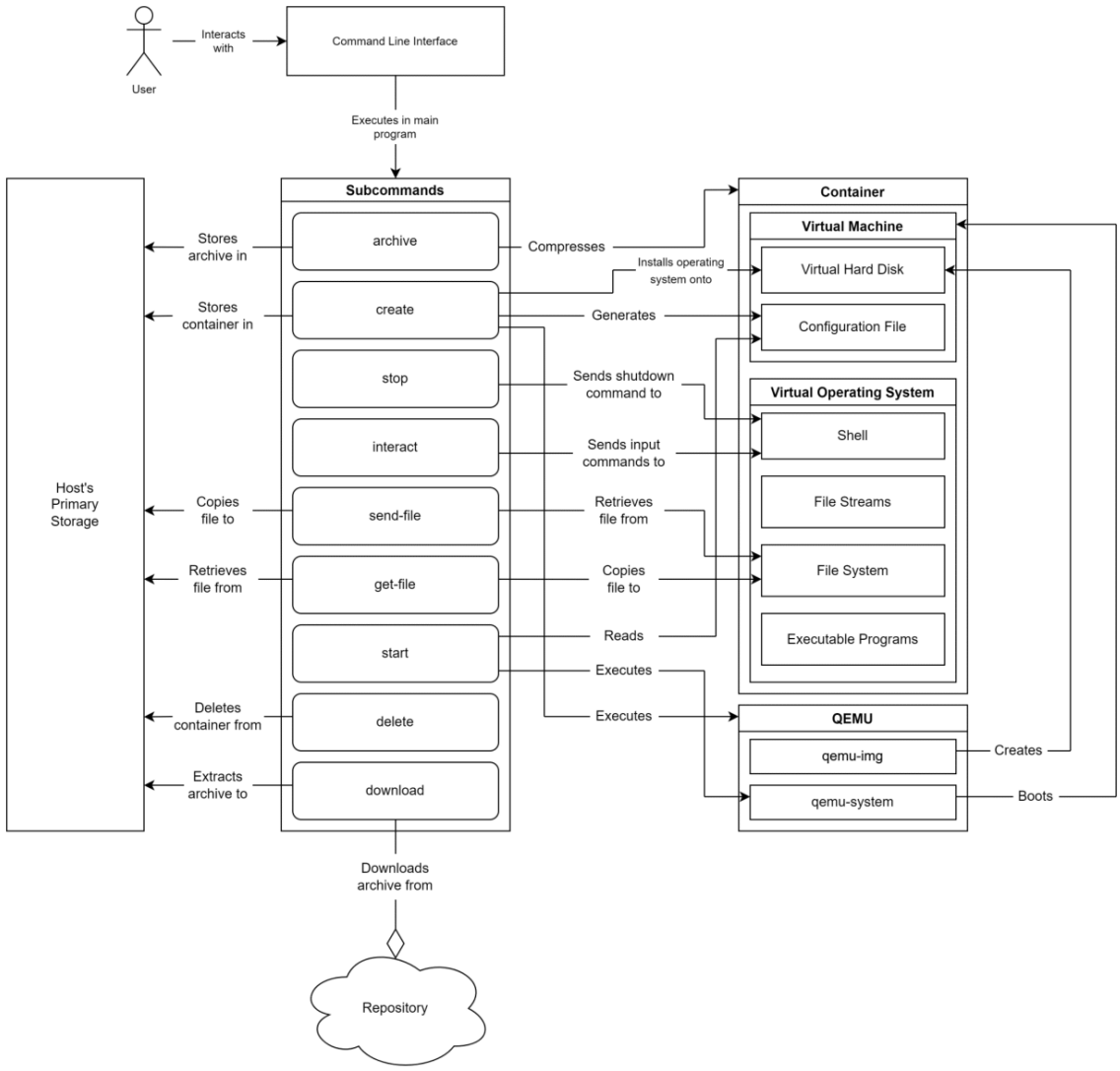
- We have created this project with an educational audience in mind, and are custom-tailoring it for specific classes at Florida Tech.
- Compared to the previous solution used by the Compiler Theory class, andrew.cs.fit.edu, our software runs entirely locally. No server component is required, which will hopefully be significantly more convenient for students and professors.

Technical challenges:

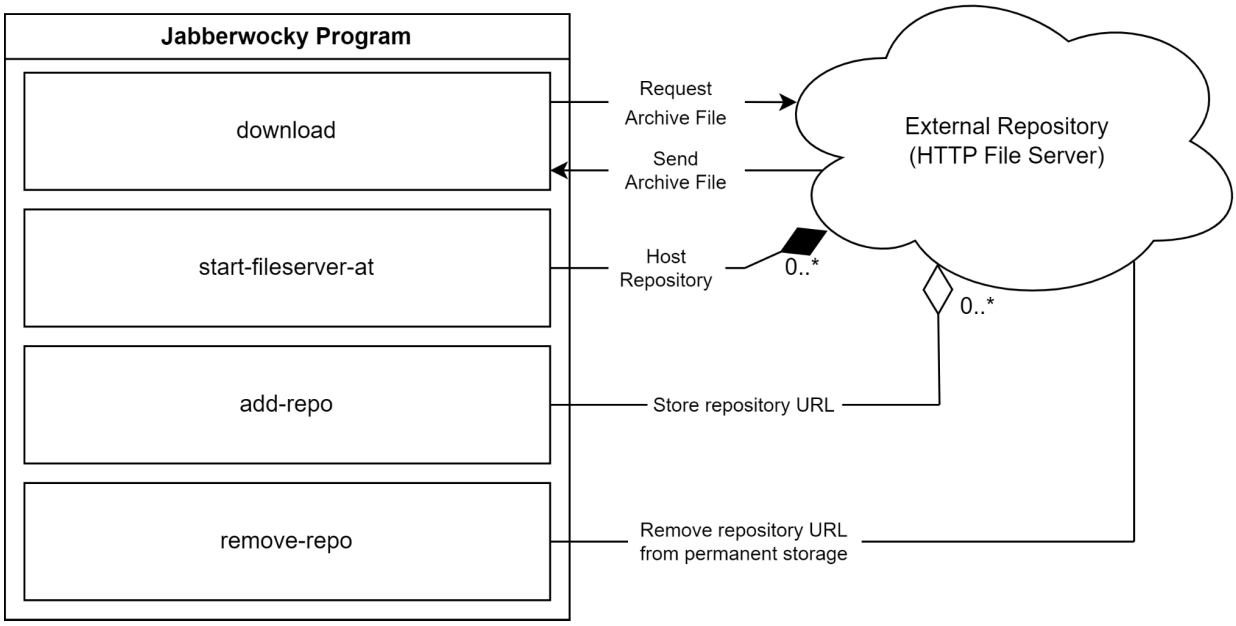
- During this semester, we plan to use feedback provided by Compiler Theory students in order to improve our system as well as to find bugs. Thus, during this process, we will need to create a way for us to easily update our system and allow the users to obtain these updates. We need to ensure that students can easily update their system to new versions without it causing new issues.
- A major task for this semester is to create a wizard that allows users to create new containers. Although we know that this is possible, we are not currently sure about the best way to implement it. We need to do more research into Qemu to determine the optimal way to automatically create new containers.
- Another major task for this semester is implementing the repository servers, which we allow users to upload and download containers from a server. We need to determine the best tool to use in order to host and dispatch these servers.

Design:

- System Architecture Diagram


User
Interacts with
Command Line Interface
Executes in main program
Subcommands
archive
create
stop
interact
send-file
get-file
start
delete
download
Stores archive in
Stores container in
Copies file to
Retrieves file from
Deletes container from
Extracts archive to
Host's Primary Storage
Compresses
Installs operating system onto
Generates
Sends shutdown command to
Sends input commands to
Retrieves file from
Copies file to
Reads
Executes
Executes
Container
Virtual Machine
Virtual Hard Disk
Configuration File
Virtual Operating System
Shell
File Streams
File System
Executable Programs
QEMU
qemu-img
qemu-system
Creates
Boots
Downloads archive from
Repository

- Repository Diagram



Evaluation:

- A major source of evaluation is to ask students to provide feedback through a survey. We will focus on surveying students who are using the tool in the Compiler Theory course. We will ask the students questions and ask them to provide a number from 1-5 for each question. The questions we will ask will include:
    - Please rate how easy-to-use this tool is
    - Please rate how likely you would be to use this tool instead of setting up environments on your personal computer
    - Please rate how easy it is to interact with a container's file system
    - Please rate how easy it is to run commands on a container
    - Please rate how easy it is to manage the containers installed on your machine (starting/stopping and installing)

- We also evaluate our tool by giving users tasks to complete and measuring the length of time it takes them to complete those tasks. Some tasks that we will ask users to complete are:
    - Create a new container with given specifications
    - Install a given container and look into its file system
    - Add source code files into a container, compile them, and then run them
- We would also like to develop a test suite for our tool to ensure that it fulfills all of our requirements. This would be implemented using Pytest. It would likely need to run the tests in some sort of Docker container to prevent it from interacting with the main system.

Progress Summary:

Module/Feature	Completion %	To do
Compiler Theory Container	95%	Work with students to improve
Operating Systems Container	0%	Deduce requirements for this container Create image that fulfills requirements
Programming Languages Container	10%	Create image that includes all needed compilers
Container Manager	80%	Add delete command Implement download, archive, add-repo, update-repo Fix bugs that occur during edge cases
Container Storage	0%	Create code for container-hosting repositories Ensure container manager can communicate with repositories
Container Creation	0%	Create wizard that converts user inputs to specifications for a container Allow containers to be created to those specifications

Milestone 4 (Feb 13): itemized tasks:

- Implement, test, and demo specifications for container creation
- Implement, test, and demo container repositories
- Implement, test, and demo more intuitive file system interaction
- Implement, test, and demo fixes found by students in the Compiler Theory course

Milestone 5 (Mar 20): itemized tasks:

- Implement, test, and demo the Container Creation wizard
- Implement, test, and demo the ability to delete and rename containers
- Implement, test, and demo Test Suite based on requirements
- Conduct evaluation and analyze results
- Create poster for Senior Design Showcase

Milestone 6 (Apr 17): itemized tasks:

- Implement, test, and demo containers for Operating System Concepts and Programming Language Concepts
- Test/demo of the entire system
- Conduct evaluation and analyze results
- Create user/developer manual
- Create demo video

Task matrix for Milestone 4:

Task	Dylan	Ian
Define specifications for container creation	90%	10%

Repositories	10%	90%
Filesystem Interaction	90%	10%
Compiler Theory Bug Fixes	10%	90%

Milestone 4 Planned Tasks:

- **Define specifications for container creation:** In Milestone 5, we want to create a standard process for the creation of containers through some sort of "wizard." We anticipate this to be a large task, so in this milestone we will first lay the foundation for it. We will decide how the end user will create a container, what options they can choose, which ones they can't, how they choose them, etc.
- **Repositories:** A key requirement of our system is the ability for users to upload and download containers from a centralized server. In this milestone, our goal is to implement the code for such a repository as well as add code to the container manager to interact with those repositories.
- **File System Interaction:** One feature our client has asked us to implement is the ability to view and modify the virtual filesystem of the container in a more intuitive way, using the host operating system's file manager program or something equivalent in functionality. Currently the send-file and get-file commands are sufficient to meet our technical requirements, but they may prove rather tedious to use with no graphical supplement. So in this milestone we will implement this feature.
- **Compiler Theory Bug Fixes:** During this milestone, students from the Compiler Theory course will have the opportunity to use our tool to complete some of their assignments.

For this reason, we want to be in close contact with these students to get their feedback so that we can quickly fix bugs and implement quality-of-life improvements.

Approval from Faculty Advisor:

"I have discussed with the team and approved this project plan. I will evaluate the progress and assign a grade for each of the three milestones."

Signature: _____ Date: _____