SCORE Milestone 3 Project Evaluation

Team Members:
Charlie Collins, ccollins2021@my.fit.edu
Michael Komar, mkomar2021@my.fit.edu
Logan Klaproth, lklaproth2021@my.fit.edu
Tommy Gingerelli, tgingerelli2021@my.fit.edu

Faculty advisor/client:

- Dr. Mohan - rmohan@fit.edu

Milestone 3 Progress

Task	Completion	Charlie	Logan	Michael	Tommy	To Do
Client-Server Implementation	90%	40%	0%	60%	0%	Handle front-end collection of arguments.
File Transfer	75%	60%	0%	40%	0%	Implement python ftp server to handle file transfers.
Auto Testing	80%	0%	50%	0%	50%	Auto Test caller, comparisons, finding files to grade.
Feedback System	50%	0%	50%	0%	50%	Implement professor-provided feedback.

Discussion of accomplished tasks:

Task 1 (Client-Server Interaction): The primary objective of this task was to split the server shell created in the last milestone into two separate applications: a front-end client shell and a back-end server shell. We then communicated between the two shells through a TCP connection. The user inputs the command into the client shell. If it is one of the recognized SCORE commands, it is sent to the server shell, where the associated module is run on the server. If it is not one of the SCORE commands, it is instead run as a local bash command using passthrough. The idea behind this is to limit the user to only sending relevant commands and not being able to send any command that may or may not be malicious. On the server side, a separate thread is spawned for each client that connects. When it receives a valid command, it runs the associated Python module. Once completed, the server will respond to the client through the same TCP connection. This response will either be a status message of whether or not the command was successful, or it will send the command output if there is any. For example, the submit command will not have any output, so the server will just respond with whether the module was run successfully. However, the view command does result in output, and so the server will respond with the output from the module.

Task 2 (File Transfer): This task involved integrating SFTP into the front-end client. This was needed as the server required the submitted files to be local, so before executing the submit module, the files to be submitted must first be transferred to the server. This is done in the front-end client after calling submit but before the command is sent to the server. This SFTP will result in the files being placed in a temporary directory in the server called /incoming. Once the submit module is executed, it will find the transferred files in the incoming directory and move them to the appropriate location. While the front-end work for file transfer is completed, there is still some work to be done in the backend. Specifically, we will need a separate application that will act as an SFTP server. This application will receive the transferred files and place them into the incoming directory. While this application will be fairly small, its main job is to manage the incoming files so that no submission is overwritten in the incoming directory.

Task 3 (Auto Testing): This task involved the addition of scripts that automatically test student submission files against professor-provided test case files. To preserve the integrity of the server and increase security, we decided to execute student files on a Docker container. We broke down auto-testing into the testing script and the script handler. The testing script handles the docker environment by modifying the Dockerfile to copy the student's submission onto the docker image. Once the Dockerfile is built, the submission file is executed in a Docker container and returned as an output log. Once the container has finished program execution, the script removes the container, and then the image is removed from the environment.

Task 4 (Feedback System): This task involves adding scripts to provide feedback to students derived by their auto testing results. This option is only available for assignments which are flagged for auto test, by the auto test flag in the assignment creation process. The current version of the feedback system is a report about the amount of passed test cases and associated feedback that would be associated with failures. This will be improved with future sprints, but is in a working state for future expansion.

Discussion of member contribution:

Charlie: I spent the majority of this milestone working on the client-server interaction. Once Michael did the initial split of the shell and got a TCP connection working, there was still a lot of work to be done to get every feature working. First, the original TCP connection would only send one byte at a time once it was figured out that the read function in Rust returns the number of bytes read. Once that was working, it was pretty simple to get the server to run the known modules when it received them over TCP. The next hurdle was getting the server to send a response back to the client. The challenge was that the server was asynchronous while the client was not. After that was resolved, I worked on the file transfer task. First, I created a file transfer function that would take a file and transfer it to an SFTP server. Then, I modified the handling of the submit command so that it would first transfer all submitted files before sending the command to the backend.

Michael: Charlie and I combined our efforts to handle the client-server process of the shell. Namely, I focused on the splitting of the previous single program shell into a server binary that receives commands from a client binary. The same python modules that were implemented for use with the preliminary shell were made in such a way that they easily integrated into the new structure of the project. Similarly to what Charlie mentioned in his contribution, we struggled a little bit with the asynchronous nature of the server and client, and handling TCP in Rust was non-trivial at first, but after some research of the documentation we found our solution to all of the problems we faced.

Tommy: I spent this milestone setting up infrastructure that could connect all of our scripts into one cohesive application rather than just a collection of programs. I developed a system of thread managers that will communicate with a main python program that I wrote. This main program will work with the host shell that Michael and Charlie developed and with the web application to queue auto tests, api calls, and auto feedback results on the systems internals. This system was a significant problem to design because it required careful thread management in order to not lose valuable job queues and to ensure we didn't accidentally block some jobs with others. A key difficulty came with designing an efficient way to sleep the main 3 sub thread managers when they weren’t in use. I made the design choice to not delete the threads because they may have

important state based information that is not returned, therefore I had to learn how to use Python Thread Events.

Logan: I spent this milestone working on a Python script that would handle the testing environment for student-submitted files. My contribution aimed to develop a script that could initialize a docker container, copy a file into the container, run the file on the container, and then return the input. I modified how I would do this multiple times, but I eventually settled on a script that would modify an existing Dockerfile, build an image with the student's submission copied in, run the image, return the input, and then delete the image. My main challenge when developing this script was balancing convenience, efficiency, and security. Despite having a way to run a file already on a docker image, I had the most trouble getting an image onto the container in the first place. I did not believe rebuilding the image with a new file for every submission would be efficient. However, after conversing with my teammates and peers, I concluded that the best overall way for convenience and security would be to rebuild the image for every submission that needs to be tested.

Task Matrix for Milestone 4:

Task	Charlie	Logan	Michael	Tommy
Finish auto testing and feedback	0%	50%	0%	50%
Front-end web development	15%	35%	35%	15%
User authentication	20%	10%	30%	40%
Server integration	50%	0%	50%	0%

Discussion of Milestone 4 Tasks

Task 1 (Finish Auto Testing and Feedback): With the end of milestone 3, auto testing and feedback was left in an unfinished state. We were able to create the functionality to spin up a Docker container which contained the user submitted files. These files are then run within these containers, and the output is piped back to the application. What still needs to be completed is the checking of this output against the test cases provided by the professor. We also need to finish the implementation of the management portion of auto testing. When a student submits a

file, it is placed in a queue. This management module will then run tests for the files in this queue when resources become available.
Task 2 (Front End Web Development): The scope of this project includes the development of two interfaces for the SCORE application, a shell and a web application. Currently all development has been towards the shell interface. However, with the server side of the application expected to be completed this milestone, we feel it is time to begin working on the web application front end. This will be done in react, and we plan to tackle most of the styling as well as routing during this milestone. This will set us up to connect this front end to a backend Node.js server that will interface with the SCORE application next semester.
Task 3(User Authentication): Currently, SCORE does not have any user authentication. This means that any user could execute any of the commands. Now that we have the application with enough functionality, we feel it is time to implement user authentication. This is vital as we only want students to be able to execute student tasks, like submitting assignments, and for professors to be able to execute professor tasks, like creating assignments. Additionally, we also need to be able to verify that users are who they say they are. To accomplish both of these problems, we are planning on implementing authentication using OAuth, along with using sessions.
Task 4 (Server Integration): With the way that milestone 3 ended, and moving into milestone 4, we will be left with several standalone features, so we are dedicating an entire task to the combination of these separate modules into one fully functional product. Namely, the Auto Testing and Feedback needs to have a standalone thread management system that needs to be integrated with the server's functionality. We plan to implement this by having a single thread in the rust server that will communicate with the thread management system written in python. Furthermore, the entire server-client system needs to be wrapped in authentication, with detailed error handling for any unexpected effect.

Dates of meetings with the client/advisor:
11/5/2024 at 11 am
11/19/2024 at 11 am

Client/Advisor feedback
Task 1 (Client-server Interaction):

- Consider making the modules only take command line arguments, this would make it far easier to send over TCP then a back and forth with user prompts.
    - Could implement the user prompts as a part of the front end.
- Make the responses that the server sends back to the front end be meaningful.
    - If a module fails, provide some explanation as to why.

Task 2 (File Transfer):

- How are you going to ensure files are not overwritten if they are all being transferred to the same directory?

    - Consider some handling on the server side so that you ensure all files have a unique name.
    - You could also consider some naming scheme in the front end to ensure this as well.

Task 3 (Auto Testing):

- One thing to keep in mind is that not every assignment can be graded simply by comparing to sample output. Some assignments will have many solutions so I use a checker instead of predefined output. How would this be used with the container? When would the output be fed into the checker?

Task 4 (Feedback System):

- Feedback will require different approaches as some programs need to be verified using a given verifier.

Faculty Advisor Signature: _____

Date: _____

Evaluation by Faculty Advisor

- **Faculty Advisor: detach and return this page to Dr. Chan (HC 209) or email the scores to pkc@cs.fit.edu**
- **Score (0-10) for each member: circle a score (or circle two adjacent scores for .25 or write down a real number between 0 and 10)**

Charlie Collins	0	1	2	3	4	5	5.5	6	6.6	7	7.5	8	8.5	9	9.5	10
Tommy Gingerelli	0	1	2	3	4	5	5.5	6	6.6	7	7.5	8	8.5	9	9.5	10
Michael Komar	0	1	2	3	4	5	5.5	6	6.6	7	7.5	8	8.5	9	9.5	10
Logan Klaproth	0	1	2	3	4	5	5.5	6	6.6	7	7.5	8	8.5	9	9.5	10