Operating System PA2 REPORT

21700337 Paek Seung Bin
21700646 Jeon Hye Won

1. Intro

A. Problem Analysis

InstaGrapd is a kind of network system for programming assignment submission and real-time evaluation. We implemented 3 modules for Instagrapd. The student submits his assignment to the instagrapd using a user interface called submitter. And to prevent for an unfortunate situation, instagrapd uses a worker to execute a student's programming assignment.

Submitter sends a student's information (ID & PW) and a c file to instagrapd. Since then, Submitter sends a request frequently for check the result of the file has been evaluated or not. And when the result of the evaluation matches the student information that submitter sent, the result is shown to the user.

Instagrapd receives student information and c file. New student information adds to the list when a new ID is entered (sign-up) and proceeds without adding a member ID to the list if the student information is not the new (login). Also, if there is an existing ID but the password is wrong, instagrapd rejects the connection. After submitter sign up or login, instagrapd pass the c file sent to the worker along with the input file in the test case directory, receive the execution result sent by the worker, compare with the output file.

Worker receives c file and input file from instagrap and tests the given c file with given input. If execution fails, send a message that failed, and if successful, send an output result to instagrapd. And if connection takes more than 3 seconds. it sends time out message. At the end of the test run, it deletes all c file and binary files for sandboxing.

B. Solution Overview

Because it was an assignment submission system for students, I thought that instagrapd should be independent and connectable from different machines or from different accounts within the same machine. Therefore, among the various IPC methods, it was considered desirable to connect using a socket. Also, instagrapd should be able to connect to multiple submitters at the same time, not just one submitter. That's why we've implemented it as a multi-threading that creates threads every time a new submitter is connected to instagrapd. As a result, several submitters were able to access instagrapd at the same time.

In addition, the instagrapd and worker must also be able to operate on different machines or different accounts too. The walker itself have to make that instagrapd does not become dangerous by executing an improper file, so it must be completely isolated in order not to affect the instagrapd, even if it goes wrong. And I thought it would be best to

implement it as a socket to communicate between these completely independent processes. When a thread is created by connecting to a new submitter in the instagrapd, the test case input and c file are sent to the worker for all given test cases, and instagrapd determines whether c file is correct or not by comparing the result value with out file. We're going to discuss into the details of this flow as follows.

The flow chart of overall program is shown below. This shows how the submitter - instagrapd - worker works when a specific user submits a programming assignment.

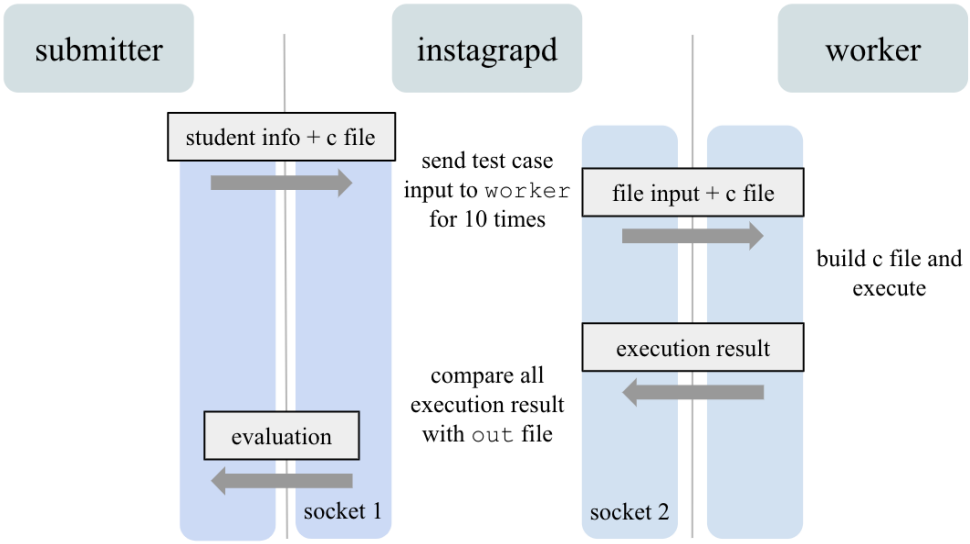

Fig #01. Flow chart of Instagrapd

2. Approach

A. Solution Design

a. Submitter

Submitter takes the input in the format given to the user, connects to instagrapd, and sends c file and student information. We used a kind of delimiter to separate student information from c files. We'll cover the details of these network protocols in the discussions. And once the submitter has sent the file, frequently send the request to the instagrapd to know whether the evaluation of the sent file is complete or not. But as we'll explain later, we haven't really implemented that part. We just keep the connection until the evaluation of file are sent back.

b. Instagrapd

Multiple submitters should be able to connect to the instagrapd at the same time on the connection between the submitter and the instagrapd, so we created the thread using `pthread_created( )` whenever a new submitter come to instagrapd. In other words, the part where instagrapd receives submitter was implemented as multi-threading. The last element of `pthread_created( )` is `void *` type, which can be passed over structure. Therefore, we put the socket number of the submitter in the structure, also the port number of the worker received when executing instgrapd,

the IP address, and the directory containing the test file. This allows instagrapd to connect to the worker as well as to the submitter in the thread.

c. worker

In order to build and run gcc from a file received from instagrapd in the worker, it needs to save the received file. And to save file, we could use a variety of approaches in generating file names, one of which is to remember the number of executions of the test cases that the worker has executed so far, and use that number as the name of the file. Using this approach will ensure that the names of the files to be executed will all be different. However, we implemented the worker as multiprocessing, not multithreading, and found that we had to use shared memory to share variables between different processes. So instead of implementing shared memory, I thought it would be better to create unique file names each time I create the file. `tmpnam( )` function creates arbitrary files name that have not been created so far. We decided that if we were to use these functions to create a filename, we would be able to create sufficiently mutually exclusive filenames.

To implement forcing shutdown after 3 seconds, we thought that signal handling would be appropriate. It was possible to create a child process and use the `sleep( )` function to shutdown the program in three seconds, but we thought that this would not be appropriate because it must takes three seconds before the process would end. The best was to hand over parameters needed for kill to signal handler function, but signal handler function did not receive parameters, so we used global variables. It was determined that there would be no problem with setting up a global variable since the process would become a independent process when it is forked.

d. Connection of various submitters in instagrapd

There are two ways to connect multiple submitters simultaneously in instagrapd, multiprocessing and multithreading. But we've implemented it with multithreading for some reasons. First, I thought that I needed to use multithreading because there was no part to be implemented using multithreading among our program. I thought that there was no reason to use multiprocessing, the task was easy, and fork makes overhead time. So we declare a global variable structure that stores user information in an instagrapd, so that even if multiple threads occur, duplication can be checked.

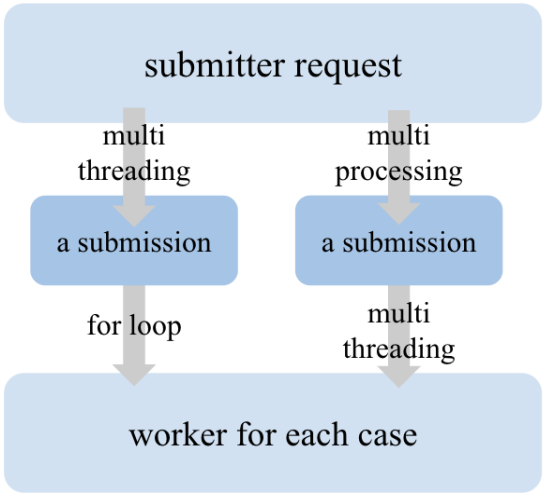

Fig #02. Difference between multithreading and multiprocessing

e. Using pipe in worker

After receiving input and c file from the worker, the gcc must be run to execute. After building, you must hand over the proper input to the executable file, save the execution result again, and hand it over to the instagrapd. We used `fork( )` and `dup2( )` to implement this method. The flow of the program is the same as the flow chart below.

First, fork the process and divide it into two, and then one process runs a gcc to check the success of the build and the execution result. After the execution result was sent to instagrapd from the parent process, by forking the process again, the c file and the binary file were deleted for sandboxing in the child process. We can just delete the file right away without fork, but we did so because we thought it would be better to do that way. More details on this will be covered in discussion.

And the executable file receives the standard input and executes the code, and sends the result to the standard output. However, executing this flow on the code required a separate process to connect the existing file descriptor to the descriptor of the file I wanted. Therefore, We implemented that the standard in connects to the input file of the executable file and the standard out connects to the out file so that the execution file I want to execute can read the value from the specific input file and save the result to a specific out file, rather than receiving the standard in and printing it out as standard out.

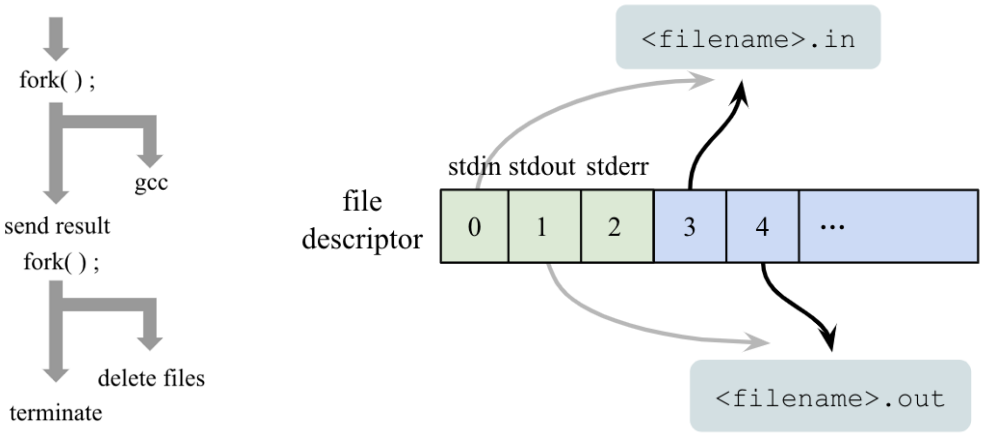

Fig #03. Explanation of worker's pipe

f. Network Protocol

The submitter sends c file and student information to instagrapd, and instagrapd send c file to Worker with a test case. We can send the sequence of characters at this time, so we need to make sure that knowing the c file start point from the receiver. There are many better and safer ways, among them, we use token. Assume of the character '@' as a token, and send data based on this token.

```
Submitter2Instagrapd          <ID>@<PW>@<C file>
    Instagrapd2Worker         @<input>@<C file>
```

3. Manual

A. Makefile

As mentioned earlier, submitters, instagrapd, and worker should all be able to run on other machines. So we created a different directory for each of the three modules, and inside it we created a Makefile for each module.

B. Execution example

Once a user didn't give anything, we printed a help screen and told a user how to give argument it. We will introduce more details and cover for various situations in the demo video.

```
OperatingSystemPA2 $./worker
Wrong Argument.
Usage: ./worker -p <Port>

OperatingSystemPA2 $./instagrapd
Wrong Argument.
./instagrapd -p <Port> -w <IP>:<WPort> <Dir>

OperatingSystemPA2 $./submitter
Please enter arguments like;
./submitter -n <IP>:<Port> -u <ID> -k <PW> <File>
```

Fig #04. Help screen with no args

4. Discussion

A. Fork or Not

As you can see in Fig #03, the worker deletes all files in the current directory for sandboxing after sending an execution result to the instagrapd. At this point, the direction we have implemented is to fork the process out of the child process and when a child process is finished normally, a parent process also terminates the worker. The process can be overridden immediately for erasing the file without fork but it is not desirable to completely replace the existing process flow, so we override another process using fork. However, there is no big problem now because of the small size of the program, but if the program grows, it might be better to replace the process immediately instead of using the fork because of the overhead time of fork.

B. How to Compare Space Character

In our program when instagrapd compares results and output, it uses simply comparing the string character by character. In here, if results have front and back blank or a white space other than space, it is wrong. In the case of the former, it can be solved by trim, but in the case of the latter, we don't know what to do. Additionally, we found a problem when executing a variety of test case. When I print results in instagrapd it is same with test output, but it prints out the wrong result in the submitter. It is difference between \r(carriage return - buffer) and \n(line feed - data). So we ignored space character when comparing output and result.

5. Limitations

A. Limitations of InstaGrapd

a. The way to submit homework that accesses to the port and IP directly can be dangerous for security reasons.
b. When you run instagrapd, you receive test case directory as instagrapd argument, where directory has input and output files for on e problem. Therefore, if you want to score c file of other problems, you must exit the instagrapd and rerun for entering the argument about test case directory. And then, we change that to a directory about another problem.

B. Limitations of Our Program

Among the PA's requirements, there is a requirement to frequently check if the evaluation result is coming after a submitter sent a file. In our program, the submitter waits to get results from InstaGrapd. However, it cannot satisfy the requirement accurately because it is not repeatedly checked. At first, we thought the solution to the requirement as follows. The student sends the c file and closes the socket. Then, make another socket and send a user ID to `instagrapd`. As a result, I predicted that it would find the result that matches the user ID sent by the second socket of the submitter and send it to the submitter. However, there were some trouble.

Because the sockets on which the submeter sends c files and the sockets on which they receive the results are different, that run on different threads. Sends the c file to the worker and receives results from the worker to the thread that was created first. So, we needed a way to connect the first thread and second thread of a submitter. At first, I thought of creating a structure that would store only IDes and the results and send the score that connects matched an ID sent by the submeter to the second socket. However, I didn't use this method because I couldn't find a way to connect to the two thread inside a submitter and when several submeters connected to Instagrapd the same time, they expected the sending messages would have twisted problem so we didn't use this method.

As a result, we connect instagrapd and submitter using only one socket. The program is implemented by closing the socket when a submitter sends after the file, waiting for the result value to come up, and receive the result value from the instagrapd and print it out.

Demo Video URL : https://www.youtube.com/watch?v=diAlJ7aDyVg