

Automatic testing of client-server applications

Helder Daniel

Departamento de Engenharia Electrónica e Informática
Faculdade de Ciências e Tecnologia
Universidade do Algarve

June 17, 2020

Abstract

Client-server application ubiquity has grown with the Cloud. Many of today apps have a client part that run in our own devices connected to a server part running in the Cloud. As with all the software, the reliability of such distributed applications is of major concern. Testing of these client-server applications can be challenging due to the need of testing not just the server and the client parts, but also both working together. This work proposes a method of testing client-server applications. To manage the submission of the source code to be tested, it is used Mooshak (Leal and Silva [2003]), an online computer programming contest manager. However the ability of Mooshak to test software is not used since it only can perform testing of solo file applications, not suitable for client-server applications.

1 Introduction

Client-server applications testing must be done executing both the client and the server applications and making them interchange messages. It is possible to test them running both in the same system or in different systems. Running both in the same system allows the testing of the software, ie it ensures that if there is no communicating network issues, the client-server application is reliable. The communication network can then be then tested on its own.

Mooshak (Leal and Silva [2003]) is a programming contests manager, designed according to International Collegiate Programming Contest rules (ICPC [2019]), that has a web portal where users can submit software to be tested and receive feedback on that testing. However it has some limitations unsuitable to test multi file applications such as client-server apps, namely it supports

only submission and testing of single file programs. Client server-application must be specified in at least 2 source code files, one for the server app and another for the client app. Furthermore clear organization of source code for real world applications often implies distributing it for many files.

Here is presented Mooshak extensions to support the black box testing of client-server applications, adding the ability to compile two applications, the server and the client, and run them together. These extensions were designed as a standalone package that can be added to Mooshak without changing a line of the original source code.

2 Mooshak testing operation

The default Mooshak's 1.x.x software testing process has 2 steps:

1. If needed, the submitted **solo source code** file is compiled or assembled first.
2. The **solo file** application is executed and tested.

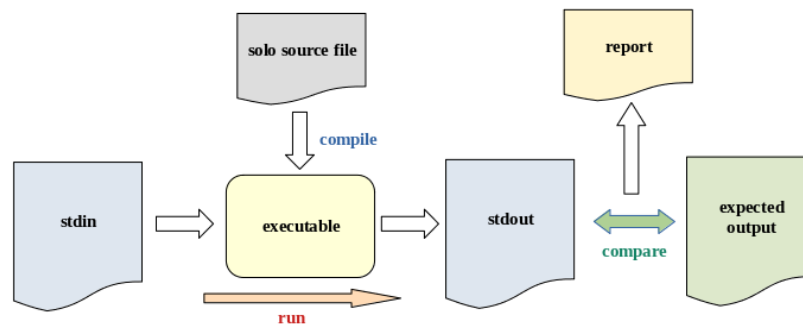


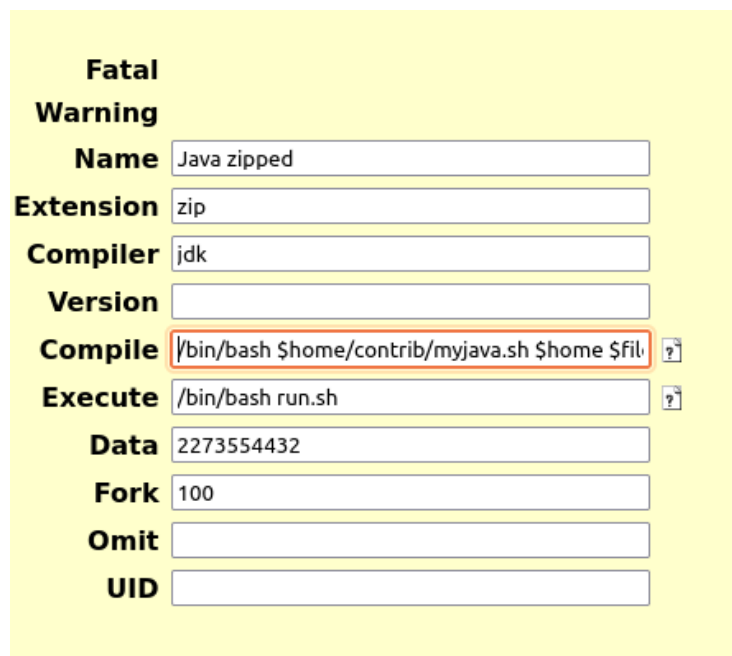
Figure 1: Mooshak default testing operation on solo files

Mooshak then performs black box software testing. Each test case has 2 files. An **input** file and an **expected output** file. The testing process, executes the application as many times as the test cases comparing the application output with the expected output, generating a report (Daniel [2019]).

3 Extending Mooshak to test client-server applications

Extending Mooshak to test source client-server application means that Mooshak must be extended to receive multiple file submissions, compile 2 applications, the client and the server and test them running together. Further these extensions were designed to be standalone and do not change Mooshak's original source code.

The submission of multi file source code is done putting all the files in a folder, packing it in a solo file and submitting this file to Mooshak. The extension unpacks it into a folder before compiling, following the same principle described in (?daniel:1). This extension is called during the compiling step, as shown in figure 1, specifying it at language settings page on Mooshak's GUI, "**Compile**" field, see figure 2.



The image shows a web form for Mooshak language settings. It has a yellow background. The form contains several fields with labels to their left. The 'Compile' field is highlighted with a red rectangle. The 'Execute' field also has a small help icon to its right. The 'Data' field contains the number 2273554432. The 'Fork' field contains the number 100. The 'Omit' and 'UID' fields are empty.

Fatal	
Warning	
Name	Java zipped
Extension	zip
Compiler	jdk
Version	
Compile	/bin/bash \$home/contrib/myjava.sh \$home \$fil
Execute	/bin/bash run.sh
Data	2273554432
Fork	100
Omit	
UID	

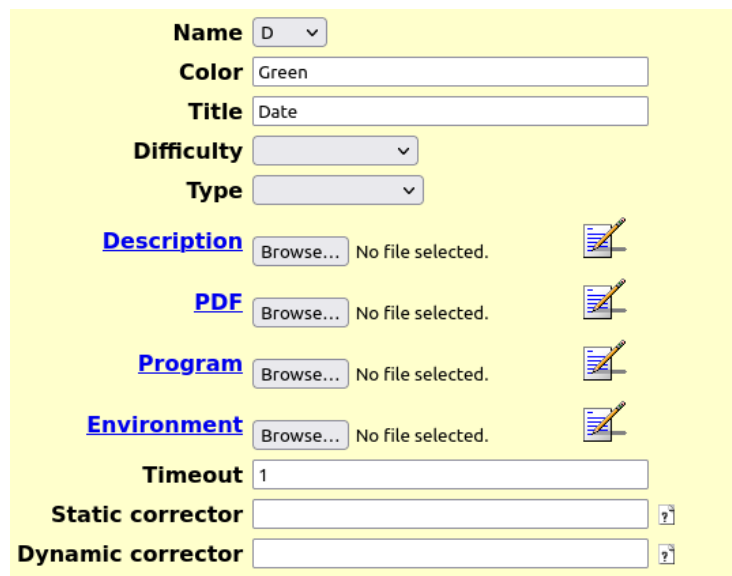
Figure 2: Mooshak language settings

At this field, and also **Execute** field, some Mooshak operation variables are available (Daniel [2019]). These variables can be passed to a script that unpacks and compiles the client-server application. This script needs information on the submission, the home directory where the submitted packed file is stored, its file name and extension. Also the \$environment variable is passed:

```
csext.sh $home $file $extension $environment
```

Note that this extension is defined in a bash script to have straightforward access to all the operating system tools available, although it was possible to define it in other programming languages.

The environment variable takes a snippet of code that can be injected along with the submitted code, to give more flexibility to the testing process, other than just black box testing. This variable can be set in the problem and test setting page of Mooshak, figure 3



The image shows a web form for configuring a problem and test settings. The form has a yellow background and contains the following fields and controls:

- Name:** A dropdown menu with 'D' selected.
- Color:** A text input field containing 'Green'.
- Title:** A text input field containing 'Date'.
- Difficulty:** A dropdown menu.
- Type:** A dropdown menu.
- Description:** A blue link, a 'Browse...' button, the text 'No file selected.', and a document icon with a pencil.
- PDF:** A blue link, a 'Browse...' button, the text 'No file selected.', and a document icon with a pencil.
- Program:** A blue link, a 'Browse...' button, the text 'No file selected.', and a document icon with a pencil.
- Environment:** A blue link, a 'Browse...' button, the text 'No file selected.', and a document icon with a pencil.
- Timeout:** A text input field containing '1'.
- Static corrector:** A text input field with a question mark icon on the right.
- Dynamic corrector:** A text input field with a question mark icon on the right.

Figure 3: Mooshak problem and test settings

3.1 Execution and testing of client-server applications

After unpacking it is need to compile the client and server application, along with the inject code through the \$environment variable, if it is present. This means that the default Mooshak testing operation in figure 1 must be surpassed, so that it is generated 2 executable files, see figure 4.

Also the server and the client must have access to stdin and stdout, meaning that the launch of the client and server applications must be tailored to accomplish it. This is performed by a generated launch and run script that allows the stdout of the client-server application can be captured and compared with the expected output, taking advantage of the default Mooshak black box testing. As stated earlier the possibility of code injection can allow some flexibility over

black box testing. To enable this operation, the launch and run script is called in the **"Execute"** field in the language settings, figure 2.

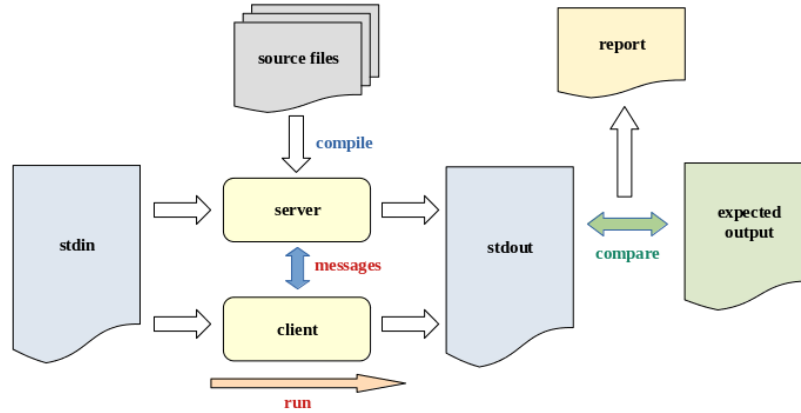


Figure 4: Mooshak testing operation on client-server applications

These extensions support testing of multi-threaded and client-server application that can communicate through common Inter-process communication (IPC) devices such as: sockets, pipes, named pipes, shared memory, or even others.

Though not designed to, with some tailoring on the generation of the launch script, even an MPI application can be tested.

There is currently a limitation though. The server and client application must be running in the same system. Currently there is no way to test an application where the server is running on a system and the client in another system. However running both in the same system is enough to test distributed software that communicates through sockets, since this communication devices can be set to transparently support communication whether the client and the server run in the same system or in different systems.

The main steps of this Mooshak extension are presented below.

Algorithm 1: Extended Mooshak client-server testing (simplified)

```

folder ← unpack(submission);
compileServer(folder);
compileClient(folder);
generateLaunchRunScript(folder);

```

To execute multi threaded and client-server application some tweaks are needed to the configuration of Mooshak's sandbox designed as **"safe execution en-**

vironment". This environment can be configured to use a limited amount of memory and launch a limited amount of threads. In Mooshak's language settings page, figure 2, field **Fork** must have the maximum number of concurrent threads.

MaxCompFork	10
MaxExecFork	10
MaxCore	0
MaxData	33554432
MaxOutput	512000
MaxStack	8388608
MaxProg	1500000
RealTimeout	60
CompTimeout	60
ExecTimeout	5

Figure 5: Mooshak all languages settings

This can also be configured on the overall language setting, figure 5, **MaxExecFork** field. Both fields specify the maximum number of threads that can be launched when testing an application, however the **Fork** field in the specific language settings, takes precedence.

4 Conclusion

Mooshak allows only the black box testing of solo file programming applications, which is unsuitable for multi file client-server application. However has the ability to receive and perform basic black box testing on source code. This extension adds support for Mooshak to receive multi file source code submissions, compile and execute multi-thread applications, as well as client-server applications. This kind of applications can be run together on the same system to perform black box testing on both. The extension is an add on, that does not change Mooshak original source code, and add the ability to test client-server applications.

References

Helder Daniel. Java unit testing framework, 2019. URL <https://junit.org>. [Online; accessed 7-June-2020].

ICPC. International collegiate programming contest, 2019. URL <https://icpc.global/worldfinals/rules>. [Online; accessed 08-November-2019].

José Paulo Leal and Fernando Silva. Mooshak: a web-based multi-site programming contest system. *Software Practice & Experience*, 33(6):567–581, 2003. URL <http://www.ncc.up.pt/mooshak/>.