System for automatic check of student solutions for Linux programming MOOCs

Mark Zaslavskiy[1,2], Maksim Kanushin[2]
[1]JetBrains
[2]Saint-Petersburg State Electrotechnical University «LETI»
Saint-Petersburg, Russia
mark.zaslavskiy@gmail.com, rextuz@gmail.com

***Abstract*—Automated laboratory and course works check for university subjects related to programming is an up-to-date task because it requires support of a big number of software environments during all period of study. Support cost for these environments grows fast because apart from keeping particular software versions, environments also should be isolated from each other and highly available for students. The work describes possible solution for automatic check of student laboratory works applied for "Linux programming basics" and "Linux Kernel Modules programming" MOOCs. The solution is a complex information system which uses virtual machines as containers for student solutions check environments. The system process student solutions for course problems which are represented as C programs and Makefiles. The check procedure includes compilation, launch and results interpretation for particular submission. During the procedure, the information system reduces mutual influence of submissions, provides unified software environment and repeatedly reproduced check results.**

Keywords—MOOC; virtualization; automatic laboratory work checking, Linux kernel, Linux programming, Vagrant, Libvirt

I. INTRODUCTION

Check automation of student solutions for practical programming tasks in IT education is an important problem because it requires continuous support of software environments for each task type during the whole period of study in university. The cost of support growth fast with number of needed environments due to complexity. Besides keeping specific software versions, each environment should provide isolation from other solutions at this environment and be highly available for students. This work describes check automation system which solves stated problem using virtualization technologies.

Proposed solution was used for checking student submissions during MOOC “Linux programming basics”[1] and offline course “Linux kernel programming”. Each task requires program development using C language and Make as a main instruments, which should do special actions defined by course authors . Checking procedure includes program compilation, execution and results interpretation. The system provides following features for the procedure:

- solution check isolation which means that any two solutions can not interfere on each other checking result;
- unified check environment for each solution;
- reproducibility of particular solution check result.

II. ANALOGS REVIEW

There is a variety of different similar online courses available in the Internet. Some of them are reviewed and compared below in order to identify key trends and common limitations of existing courses related to Linux kernel programming and Linux system programming. These two topics are analyzed together because different authors gives overlapping definitions for these two terms.

The review compares only online courses despite there are a plenty of different online services allowing to evaluate code quality or perform a check of program behavior using stdin or command line arguments. These services were excluded because they provide less plausible environment for OS-specific tasks by giving a very limited sandbox.

A. Description of courses

A to Z of Linux System Programming [2]

The course allows the participant to develop a deep understanding of Linux or Unix systems and learn concepts and skills which are essential for system programming and software development on Linux-based devices.

Linux Device Drivers: Programming at the Kernel Level [3]

The course covers basic Linux kernel programming from the very beginning explaining what device drivers are, how kernel modules work and how to debug the code. The course provides a student with video lectures and files such as hardware emulator and example code to try the material on a local machine.

Linux System Programming [4]

The Linux System Programming online course introduces the student to Linux system programming fundamentals and

teaches him how to write programs which interact with the Linux kernel.

An introduction to Linux kernel programming [5]

This course is a multi-week, online technical course which contain a series of lessons to introduce the student to the basic Linux kernel programming.

Linux Kernel Internals and Development [6]

This course is held by the Linux foundation. It helps the students to learn how to develop for the Linux kernel. It provides information about Linux architecture, what are the basic methods for developing in the kernel, and how to cooperate with the Linux developer community.

Linux Programming and Driver Developer [7]

An expert level course covering multiple aspects of programming on Linux including system programming, kernel programming and a lot of attention is paid to device drivers development as well.

B. Criteria

1) Topics

Since the proposed solution is used for MOOC "Linux programming introduction" and offline course "Linux kernel programming", the topics reviewed were "Kernel programming" and "System programming" to match the topics used in our solution. Kernel programming and system programming use two different approaches to programming for Linux. First one suppose writing modules for the kernel which have little to no interaction with the userspace. System programming, on the other hand, covers only programming in the userspace without affecting the kernel anyhow.

2) Exercise format

Format of exercises can vary from course to course. Some courses use simple tests or quizzes, some require installation of corresponding packages or even whole operating system on a local machine and some allow to use just a browser to check the solution remotely.

3) Free or paid

Price for the participation in the course can differ drastically. Some courses are free, some are paid or require an active subscription for the MOOC they belong to.

4) Number of hours

Number of hours that a course take is not always connected with the course's price. Finding the optimal length of the course and it's connection to the course's contents is an important task and is worth to be researched.

5) Difficulty of deployment

For the courses which require a setup on a local machine it's important to know how simple or complicated the installation procedure can be, for other courses this criteria does not apply.

6) Availability of the platform extension

Most courses turned out to be non open source and the criterion could not be applied to them.

TABLE I. COMPARISON OF LINUX KERNEL AND SYSTEM PROGRAMMING COURSES

Course name	Topics		Exercise format			Free or paid	Number of hours	Difficulty of deployment
	Kernel	*System programming*	*Quizzes*	*Execution*				
				Remotely	*On a local machine*			
A to Z of Linux System Programming	-	+	+	-	+	paid	15	easy
Linux Device Drivers: Programming at the Kernel Level	+	-	-	-	+	paid	6	hard
Linux System Programming	-	+	+	-	-	free	12	-
An introduction to Linux kernel programming	+	-	-	-	+	paid	30	easy
Linux Kernel Internals and Development	+	-	n/i	n/i	n/i	paid	12	n/i

According to the Table 1 (n/i means "no information") there are no online courses about Linux programming which allows to execute and check task solutions remotely. Instead the most common approach is a setup of check environment on a student computer which makes learning materials less accessible, may lead to non-unified checking environment and potentially does not isolate solutions properly. Also all courses provide only one topic: kernel or system programming.

Review results shown that existing courses does not provide any systems that can be adapted in order to create remote solution verification backend. Limitations described above can be solved by developing a new checking system

which uses virtualization technologies because VMs provide isolated and scalable virtual environment for student solution behavior verification. Architecture of the checking system is given below.

III. Architecture

The system consists of two interrelated components – web-application and daemon. Student solutions are submitted from outside by special HTTP requests which are parsed by web-application and transferred to daemon for further check. Results become available using web-application REST interfaces.

Solutions checking procedure is consequent: current solution will be processed only after the previous one. Also, all incoming submissions will be ignored if the system already has a solution to check. Such architectural pattern was chosen in order to minimize risk of parallel execution errors. To have an ability to check tasks simultaneously system can be horizontally scaled with an appropriate proxy setup.

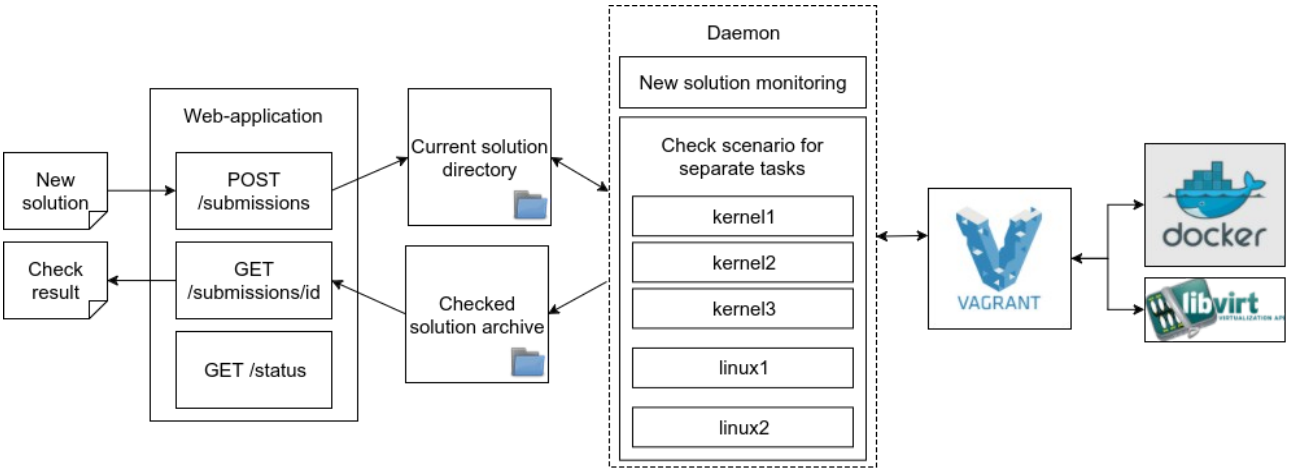

Fig. 1. System architecture

Web-application for student solutions receiving was written using Python and Flask [8] framework, which performs HTTP requests parsing. Flask was chosen because this library allows to setup a lightweight and reliable web-application quickly. Application is connected to Apache web-server using WSGI [9] protocol. Basic REST interfaces:

1. POST /submissions. This request allows to add new student submission to system. Input data is formed as a JSON containing source code, makefile and task identifier.

Depending on the server current state them system can answer in two different ways: it can return unique submission identifier (submission_id) or it can return error message in case of current state does not imply new submissions. Received submission is transferred to daemon using local file system.

2. GET /submissions/<submission_id>. Following request provides interface with read-only status of particular submission check, also containing execution and build logs.

3. GET /status. This request prints current server status as a string.

Web-interfaces are not protected by any authentication mechanisms because the system is designed to work as a part of Stepik platform which preforms its own authentication.

Checking daemon is implemented as a set of Bash and Ruby scripts. Bash scripts are used to ease the access to Linux commands and Ruby was chosen since Vagrant itself is written in it which allows a custom Ruby to access Vagrant with ease. Checking daemon performs constant monitoring of student solutions from the web-application. After receiving new submission daemon performs following actions:

1. Static solution sanity check.
2. VM start.
3. Copy of solution files to VM.
4. Solution compilation and dynamic check.
5. Check artifacts collection.
6. VM stop.
7. Check procedure status decision making.

For satisfying minimal performance and security requirements daemon limits checking time and interprets overlimit as automatic check fail. Passing of each checking algorithm step is accompanied with server status change (Fig. 2) which is available by /status REST interface. Possible status values are:

- IDLE - server is ready for new submissions,
- VALIDATION - static check of solution,
- VM_START - system is starting VM,
- BUILD - system is building and checking solution,
- VM_STOP - system is stopping VM,
- VERDICT - system makes decision about check status,

- TIMEOUT – check duration reached limit.

Already checked submissions are stored on a local file system in special directory.

Check procedure is aimed to verify student solution behavior and due to this fact there are potentially infinite set o f correct solutions. Despite that formally check results is binary, students get additional output from checker with detailed description of particular check stage were solution failed desired behavior. Also there is a probability that solution check will result type one or type two error because of checking system failure. For this case system provide special "Fail reason" field in /status response.

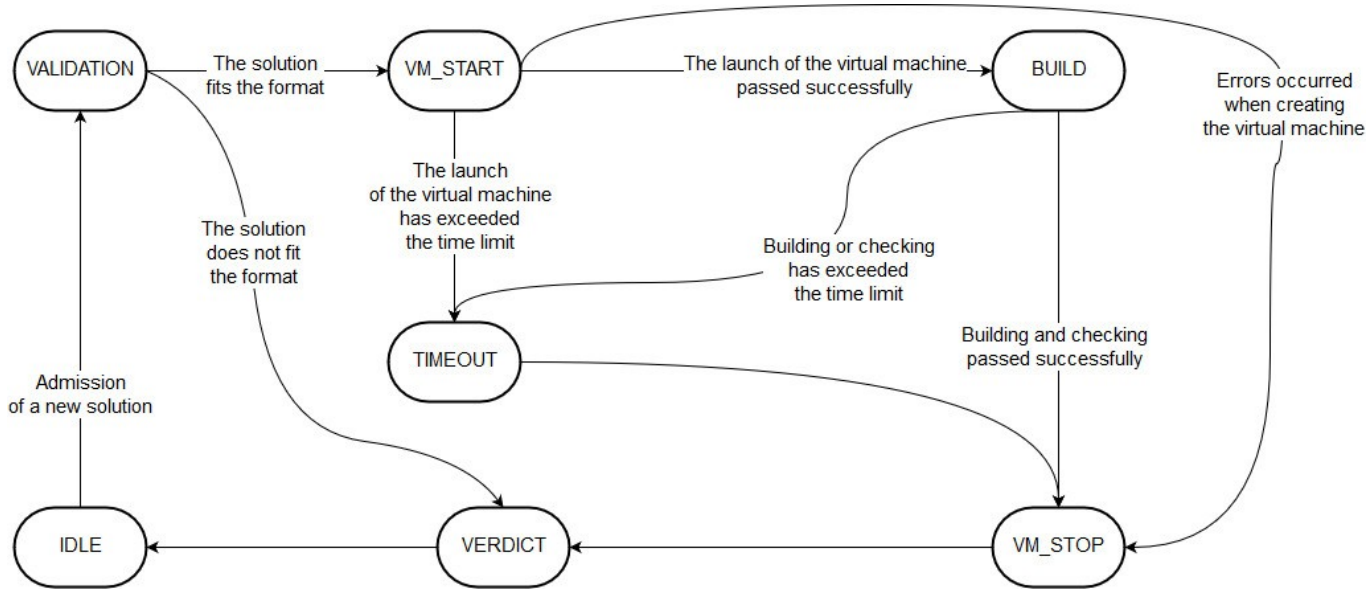

Fig. 2. Solution checking states graph

IV. Virtualization technologies

Vagrant [10], a tool for creation and management of virtual machines, is used for student's solution check. This tool allows to disengage from the specifics of virtualization technologies and provides a uniform interface for virtual machines control. Docker [11] technology is used to verify solutions which do not require Linux Kernel modification, as it provides sufficient level of environment isolation and a high level of performance. Libvirt [12] is used as a provider for those solutions which require Linux Kernel to be modified.

A. The format of the test scenarios

Check system is organised in a modular fashion - it allows to work with any number of tasks and do not have any strong connection with them. Every task in terms of the system represents a set of scenarios which perform launch and configuration of a virtual environment as well as building and verifying of the student's solution. For the successful solution checking, test scenarios must contain a Vagrantfile with the definition of the virtual environment and a Ruby scenario for the build and the solution checking itself.

The system allows the following ways for checking within the solution: comparing the student's solution with the reference one by the stdout content or using a custom bash-script which will resolve the compliance with the task. The system acquires the solution checking result implicitly by capturing the console output in the virtual machine and explicitly through the status.txt file with the binary solution checking results, which must be generated by the scenarios inside the virtual machine.

B. Implementation of checking for Linux kernel tasks

A student who solved the tasks has to send a kernel module [13] written by him and a Makefile [14] to build it. After the system receives the solution it loads it into a separate Libvirt virtual machine where it is built and loaded into kernel. At this point the solution checking itself happens alongside checking the validity of building the solution, its loading and unloading from kernel.

Checking procedure described below was used for implementing real kernel tasks, which will be part of "Linux kernel programming" MOOC first launch:

- simplest loadable kernel module,
- intermodule function call,
- kobjects[15] creation and manipulation,
- kernel module parameters processing [13],
- character device drivers [13]:
  - singleton device,
  - IOCTL implementation,
  - dynamic device node creation.

The student's solution checking consists of several steps:

1) Environment preparation

For proper building of student and service kernel modules following packages should be installed inside the VM:

- gcc [17];
- Make;
- linux-headers [18], header files for using kernel API of the VM.

In the current version of the system problem of environment uniformity is solved by using a special Vagrant box with already installed dependencies. The box is set as a base image for any VM for kernel tasks started by system. Student solution, Bash scripts for solution checking and service kernel module (depends from task type) are loaded inside the VM in case of successful start. Before running any solution checking procedures system performs compilation and loading of a service module and removes its sources for avoiding reverse engineering .

2) Load of student solution module

Student solution is compiled and loaded only after all environment setting procedures finished. If there are any errors, the solution checking procedure is terminated and its result is marked as failed. Logs of executed commands on the loading state are saved including both stderr and stdout and become available to student.

3) Retrieving check result of student module

After loading into VM kernel student module should do special actions defined by task. Monitoring of the module is done by Bash script and service module. After receiving student module work results the script make decision about observance of conditions. Particular methods of solution checking differs from task to task but the basic checks are exit code values control, generated kobjects behavior check and syslog content mining [16].

4) Modules unload

After finishing task conditions check the system unloads modules starting from student one. Success of this operations is checked by exit code because solutions should not only perform all needed actions but also perform cleanup operations in a correct manner due to higher coding standards for kernel extensions.

V. CONCLUSIONS

The developed system allows to reduce price and provides isolated environment for student solution check for the Linux programming subjects. The part of the system that was designed for checking student solutions for the “Linux programming basics” course have already been released and has successfully worked with over 6500 students uploading and checking their tasks. Current realization supports only programs written in C language, but such format is applicable for a vast set of different topic from general programming to kernel modules development. Future plans:

- interfaces for using different programming languages and software environments;
- system approbation during first launch of the “Linux kernel programming” MOOC.

REFERENCES

[1] Основы программирования для Linux. // Stepik URL: https://stepik.org/course/Основы-программирования-для-Linux-Linux-548/ [Accessed: 28- Apr- 2017]

[2] A to Z of Linux System Programming // Udemy. URL: https://www.udemy.com/a-to-z-of-linux-system-programming/ [Accessed: 28- Apr- 2017]

[3] Linux Device Drivers: Programming at the Kernel Level // Gogo Training. URL: https://gogotraining.com/training/courses/254/linux-device-drivers-programming-at-the-kernel-level/ [Accessed: 28- Apr- 2017]

[4] Linux System Programming [LSYS] // VirtQ. URL: https://www.virtuq.com/module_details/linux-system-programming [Accessed: 28- Apr- 2017]

[5] An introduction to Linux kernel programming// CrashCourse. URL: http://www.crashcourse.ca/introduction-linux-kernel-programming/introduction-linux-kernel-programming [Accessed: 28- Apr- 2017]

[6] Linux Kernel Internals and Development // Linux foundation. URL: https://training.linuxfoundation.org/linux-courses/development-training/linux-kernel-internals-and-development [Accessed: 28- Apr- 2017]

[7] Linux Programming and Driver Developer // EDA solutions. URL: http://techveda.org/linux-programming-and-driver-developers/ [Accessed: 28- Apr- 2017]

[8] Flask. URL: http://flask.pocoo.org/ [Accessed: 28- Apr- 2017]

[9] Gardner J. The web server gateway interface (wsgi) //The Definitive Guide to Pylons. – 2009. – P. 369-388.

[10] Vagrant by HashiCorp. URL: https://www.vagrantup.com/ [Accessed: 28- Apr- 2017]

[11] Docker. URL: https://www.docker.com/ [Accessed: 28- Apr- 2017]

[12] Libvirt virtualization API. URL: https://libvirt.org/ [Accessed: 28- Apr- 2017]

[13] Corbet J., Rubini A., Kroah-Hartman G. Linux device drivers. – " O'Reilly Media, Inc.", 2005.

[14] Salzman P. J., Burian M., Pomerantz O. The linux kernel module programming guide //TLDP: http://tldp. org/LDP/lkmpg/2.4/html. – 2001. – P. 82.

[15] Mochel P. The sysfs filesystem //Linux Symposium. – 2005. – P. 313.

[16] Yamanishi K., Maruyama Y. Dynamic syslog mining for network failure monitoring //Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining. – ACM, 2005. – P. 499-508.

[17] GCC, the GNU Compiler Collection. URL: https://gcc.gnu.org/ [Accessed: 28- Apr- 2017]

[18] Package: linux-headers-amd64 (3.16+63).. URL: https://packages.debian.org/jessie/linux-headers-amd64 [Accessed: 28- Apr- 2017]