RESEARCH ARTICLE

CLIK: Cloud-based Linux kernel practice environment and judgment system

Hyunchan Park1 | Youngpil Kim2

1Division of Computer Science and Engineering, Jeonbuk National University, Jeonju, Korea
2School of Computer Science, Semyung University, Jecheon, Korea

Correspondence
Youngpil Kim, School of Computer Science, Semyung University, 65 Semyung-ro, Jecheon-si, Chungcheongbuk-do 27136, Korea.
Email: ypkim@semyung.ac.kr

Funding information
National Research Foundation of Korea (NRF), Grant/Award Numbers: NRF-2017R1C1B5016000, NRF-2020R1F1A1067365, NRF-2020R1G1A1005544; Jeonbuk National University, Research Base Construction Fund Support Program in 2020

Abstract
Assignments on kernel programming are essential parts of operating system (OS) courses taught to computer science students to provide them a deep understanding of real-world OSs. However, these assignments require tremendous effort from both students and instructors. Students are routinely flustered by the daunting task of building a practice environment from scratch; instructors are pressurized for time while validating a student's work that requires several kernel installations and reboots. To minimize this effort, we propose CLIK, a cloud-based Linux kernel practice environment supporting automatic judgment. It provides students with an individual and easy-to-use kernel practice environment and instructors with a fast and easy evaluation of students' work with live feedback. Our experiences with two assignments from a real-world OS course carried out on CLIK show that CLIK can successfully provide Linux kernel environments for 40 students and help instructors greatly by validating their kernels within one minute using parallel and automated judgments. We also describe detailed lessons learned from developing CLIK that will help both researchers and instructors building similar systems.

KEYWORDS
automatic judgment system, cloud system, computer science education, Linux kernel assignment, operating systems

1 | INTRODUCTION

Operating system (OS) courses need to cover both theoretical and practical material because understanding OS abstractions requires knowing how to hide low-level details consisting of various hardware features, knowing how to organize the feedback from these low-level systems into high-level theoretical perspectives, and knowing how to deliver these perspectives to users or other subsystems. Therefore, OS courses need undergraduate students to gain both theoretical and practical knowledge and to use this knowledge proactively in as many programming assignments on the real kernel source code as possible. For example, for a good understanding of process scheduling, in addition to knowing how to choose the next process, students need to know how to exchange the hardware context of two processes. If the target OS is based on a Linux kernel, the specific implementation code will reside in the “/arch” subtree.

In an OS course, there are generally three kinds of programming assignments: (a) real-world OS kernel code, (b) tiny experimental/educational OS, and (c)
user-level pseudo-OS. Compared to the others, assignments using the real-world OS kernel code have several benefits. First, students can obtain a deep understanding of a real OS. The OS kernel contains all of the subsystems covered by an OS course, and these subsystems all work in the practice environment just like they do in the real world. In contrast, a user-level simulator or an educational OS only provides a few abstracted versions of subsystems. Second, students who have already worked on the real-world OS kernel code will have practical experience that will help them to take up related tasks immediately after graduation. The OS kernel code is large and highly complex, it requires a considerable amount of time to understand and modify correctly. Therefore, it is beneficial for the students to gain experience of this during their studies under the guidance of an experienced instructor, hopefully making them confident in tackling related tasks after graduation. Third, by working with real-world OS kernel, students will have experience with several useful services that are implemented and utilized in the OS kernel. The Linux kernel has lots of exemplary codes making it quite the most useful toolbox for students. For example, the “List” service of the Linux kernel is particularly useful to implement other data structures such as bidirectional queues, stacks, and several trees including the radix and red-black. Given the reasons stated above, we strongly believe assignments on a real-world OS kernel are best for students.

In practice, however, there are several difficulties with assignments on a real-world OS kernel for both students and instructors. For students, it is hard to equip the environment to build the OS kernel. The building process of the OS kernel source code requires many resources, including central processing unit (CPU), memory, and storage. For example, the building process of the Linux kernel source code requires several hours when run by the desktops or laptops that are commonly possessed by students. It becomes difficult for the students to complete their assignments if the debugging process takes such a long time. In addition, the various system failures caused by kernel errors require rebooting of the entire computer system, making debugging even more difficult. As it is impossible to provide a high-performance computing system for each student, our study focuses on providing an effective and easy-to-use OS programming environment for students.

Instructors face different kinds of difficulties when it comes to evaluation and, at the moment, the task is time-consuming and requires much effort on the lecturer’s part as they need to run and test students’ programming assignments many times to check that they are working correctly and whether they satisfy the requirements set. Notably, it takes several minutes to build an OS kernel programming assignment even in high-end servers. Furthermore, not only does the build time interfere with the instructor’s evaluation but also the constant need to reboot their machine is a time sink. After building the OS kernel source code, rebooting is required to validate the student’s kernel——this is a direct result of this kind of assignment. Therefore, the difficulties in working on and evaluating OS kernel assignments are many and varied for both students and instructors; thus, we set the task of overcoming these difficulties as our main goal.

To achieve this goal, we suggest CLIK, which provides both a cloud-based practice environment and an automatic and parallel judgment system for Linux kernel programming. In terms of an OS assignment environment, CLIK has two benefits: First, CLIK provides a rapidly deployable OS assignment environment based on cloud computing and supports virtual networks; therefore, it can easily support various network configurations. In addition, CLIK reduces the cost of the installation and setup of an OS and hardware by supporting various instance types and images. Second, CLIK provides an automatic OS assignment evaluation platform. Moreover, it provides both instructors and students with the ability to access automatic judgment of the submitted work, this results in efficient evaluation of large amounts of kernel programming assignments. We have used CLIK for two essential kernel assignments in our OS course: inter-process communication and synchronization. Our experiences with CLIK have proved that it is beneficial to all and has great potential for future use.

The rest of the paper is structured as follows. Section 2 provides our design principle with explanations of key concepts. Section 3 describes real-world practices of CLIK for representative OS kernel programming assignments with practical results. Section 4 shows comprehensive reviews of previous studies. Finally, Section 5 concludes the paper.

## 2 DESIGN

Our goal is to provide a convenient computing environment for both instructor and student; a further aim is to provide easy assignment set in our environment which is also easy to rapidly evaluate in a mass. The key to this is automatic and parallel judgment. To realize this automatic judgment for kernel programming assignments, we designed and implemented CLIK.

Figure 1 shows the overall architecture of CLIK. CLIK consists of three main parts: human users, an automatic judgment system, and a cloud system. First, CLIK supports two types of users: instructors and students. The instructor announces kernel programming assignments to students and submits the correct version of the kernel
(i.e., the instructor-made solution kernel) to CLIK. After announcement, students try to complete their assignments and submit their test version kernels (student-made kernels) to CLIK.

The second part is the CLIK cloud that consists of a set of virtualized computing resources. In the cloud, users can develop, modify, and execute various versions of custom kernels like on a bare physical machine. CLIK uses a cloud system to share a high-performance server with the students to deploy the practice environment more easily. A student obtains an instance, which is a virtual machine deployed on the cloud, on which to perform their OS kernel assignment. An instructor also obtains an instance on the same virtual network as the students, the instructor’s instance is used to judge the student’s outputs automatically by accessing the student’s instances.

The third part is the automatic and parallel judgment system. CLIK evaluates the two versions of the submitted kernels in the judgment system, that is, the instructor-made kernel and student-made kernel. CLIK first generates a correct version for the test cases using the instructor-made kernel and, then, evaluates the results of the student-made kernels by comparing them with the correct results. CLIK summarizes the evaluation results automatically into digital reports. However, these reports are delivered to users selectively. For example, the instructor receives a full version of a report, whereas students receive limited reports to avoid revealing sensitive data. Note that the evaluations of student-made kernels are performed in parallel to speed up the evaluation procedure.

Now, we try to justify our design choices based on two points. The first point explains why and how we adopted a private cloud system as an OS course assignment platform. The second point explains our key software and the automatic and parallel judgment system in more detail.

2.1 Deployment on cloud computing

2.1.1 Practice environment as a service

CLIK is based on a cloud system that is cost-effective and suitable for large-scale and parallel computing tasks such as artificial intelligence, Internet of things, and fog/edge...
computing \[^{5,10,13}\]. Especially, CLIK is built with OpenStack \[^{15}\], the most popular open-source project for creating cloud systems. Owing to free and powerful software tools such as Apache CloudStack \[^2\] and OpenStack, nowadays, building a private cloud system is easily accomplished by a department or college. If there were difficulties gathering the hardware or human resources for such a task, a well-managed commercial public cloud system such as the Amazon Web Service (AWS) \[^3\] or the Google Cloud Platform (GCP) \[^9\] can be used. AWS and GCP both provide plenty of support including free credits for students and instructors of educational institutes. CLIK is not limited to private cloud systems; it can easily be used on commercial and public cloud systems as well. Thus, any difficulty in building or utilizing the cloud system required for CLIK should be easily overcome.

These kinds of cloud systems satisfy several requirements as a practice environment for OS assignments. First, isolation between students can be accomplished by using the functionality of network security. For example, we can prohibit network connections between instances and isolate storage systems via virtualization. Thus, manipulation of or sneaking into other students' instances or storage systems is not possible.

Second, access control can be easily made and tightly enforced among components such as a network and a snapshot. CLIK gives the highest privilege to the instructor so that the instructor can log on to an instance of a student via the Secure Shell (SSH) protocol. Also, we should consider the use of a snapshot of an instance. The snapshot is a complete replication of an instance. So snapshot is very useful for students in terms of backup of programming work because students can restore their work safely from a snapshot when their instances do not respond due to the incorrectly modified kernel. However, the snapshot can be maliciously used if it is allowed that some students copy or manipulate other students' snapshots. To prevent it, we only allow students to resume or execute their instance and prohibit them from creating a new instance using the snapshot. Especially, if only a student legally requests for creating an instance using a snapshot, an instructor handles the student's request manually.

Third, the cloud system gives a high level of accessibility, enabling students to work on their assignments anywhere they want, including from computing rooms in schools or their homes. Moreover, the instructor can easily access the student’s environment to help or score their work. Thus, by simply using a cloud system, we can fulfill the essential requirements for programming best practices. Although we did not directly contribute to cloud systems and the benefits they give, CLIK inherits those benefits simply by being a cloud-based system.

2.1.2 Virtual network

We create a virtual private network with a gateway for internet connections. The gateway has a fixed public internet protocol (IP) and users can connect to the private network using this IP. The network allocates private IPs for instances using the Dynamic Host Configuration Protocol. CLIK uses the port-forwarding facility of the gateway for delivering the packets from the internet to the designated instances. Figure 2 depicts the CLIK network topology with several instances as an example. The internal private network uses the CIDR configuration with an IP address “192.168.0.0/24” so that about 253 IP addresses can be allocated to the instances. The router is allocated a fixed IP address of 203.254.143.169. In OpenStack, the router is implemented by a Linux bridge and the Network Address Translation function of the routing table can be used for port-forwarding of the Linux bridge. The router forwards packets for specific ports to the SSH ports of instances. For example, a packet delivered to port 19041 of 203.254.143.169 will be forwarded to 192.168.0.41:22, where 22 is the default port number for SSH. Figure 3 shows an example configuration of iptables providing this routing facility in Linux. In this example, we use the port number 7777 for SSH connection due to security reasons. It is widely recommended to use a modified port number instead of the default port number as this is well known and so is exposed to intrusion.

The network configuration of CLIK provides three advantages for computing resource management. First, the CLIK network supports a port-forwarding facility by an internet gateway and provides students with a mechanism to access even instances that do not have public IPs. Second, the CLIK virtual private network helps save public IPs, a limited resource. As the OS is an essential component of any computer science degree, many students choose this course. In most cases, a private cloud that is run by a department or a college cannot provide a public IP to each of the students. CLIK overcomes the problem of limited public IPs as it only needs a single public IP for a shared Internet connection. Third, the CLIK gateway acts as a firewall, preventing malicious attacks as remote access is denied. If more security is needed, it is straightforward to configure a routing table, thereby denying access to unknown IPs. Although this requires registration of known IPs in the routing table manually, if there are many attempts to break into the network, the risk is considerable and this is a small price to pay.

We summarized three steps and their security effects for configuring virtual network in terms of security process in Table 1.
After creating a network, we define specifications for instances. A predefined specification in OpenStack is called a flavor. The allocated specifications should be carefully decided considering both the total capacity of the resources in the cloud system and the minimum requirement for each assignment. For example, we provide 12 virtual CPUs (vCPUs), 4 GB memory, and 40 GB storage for a student’s instance in our experience. To provide the flavor for 100 students, we need 1,200 CPUs, 400 GB of memory, and 4 TB of storage. However, these requirements do not have to be provided by the physical servers because the physical resources are virtualized.

| Steps                  | Target                  | Results                          | Security effects                      |
|------------------------|-------------------------|----------------------------------|---------------------------------------|
| 1. Creating a virtual network | Network connections      | Generating internal network      | Isolating of an internal network      |
| 2. Allocating private IPs with DHCP | All instances           | Assigning instance’s private IP  | Identifying an instance               |
| 3. Port forwarding      | Malicious network access| Access control by a gateway      | Preventing the well-known port attacks|

Abbreviations: DHCP, Dynamic Host Configuration Protocol; IP, internet protocol.
OpenStack provides an over-provisioning functionality that can allocate more virtual resources than physical resources. In the case of CPU, OpenStack can allocate 16 times more vCPUs than physical CPUs (pCPUs) in the default setting. The memory can be allocated for instances 1.5 times than physical memory. Our OpenStack system consists of one server as a controller node and three servers as compute nodes. The compute nodes equip 132 pCPUs and 576 GB memory. Thus, 2,112 vCPUs and 864 GB memory can be allocated which is enough for 100 students. Also, dozens of solid state drives (SSDs) are configured as the RAID5 storage system with 8 TB capacity and shared across the nodes. In our experience, our computing resources were enough to support more than 100 students so that the numbers will be a good guide for the other instructors.

Then, we prepare an OS image, this usually contains a virtual storage device with a bootable OS for creating the instances. When preparing the image, we consider several tools including compiler and linkers that need to be preinstalled. Our goal is to enable students to gain experience in kernel compilation and installation so that they can handle other trivial things such as installing compilers and setting up a method for file transfer. In cases where students do not have any experience working with Linux, an instructor can provide a fully prepared OS image to those students, making working on a given assignment less daunting. However, we believe such beginners will not be able to explore and gain the required knowledge following this approach.

In our first experience with CLIK, we decided to provide an image with only an OS. We used the Ubuntu cloud image with two modifications: the SSH port number and a preinstalled RSA key for SSH access of the instructor.

Finally, students can deploy their instances using the flavor and provide images on the virtual network whenever they want. Due to limited resources, we suggest each student be limited to using one instance at a time. However, this is not mandatory because when a student creates a new instance, he or she may want to copy some files from the old one. According to our experience, the total usage of instances was not over our expectations even with a loose policy.

2.2 | Automatic and parallel judgment

CLIK provides an automatic and parallel judgment system for both students and instructors. For the instructor, an essential feature of the judgment system is to score numerous students autonomously. The CLIK judgment system also generates well-organized reports, including student ID, test results, submission date, and other necessary information for the instructor. Furthermore, the judgment system can provide students fast feedback from an instructor about the current assignment. This timely feedback is very beneficial for students because they can rectify their mistakes and any underlying misunderstanding about the purposes or requirements of the assignment, see their progress with the course material, and get an estimate of their score while the subject is fresh in their memory. Feedback to a single student is not draining; however, giving intensive feedback to every student is very challenging because it is time-consuming, and instructors usually do not have enough qualified assistants. The automatic judgment system solves this problem because all students can pre-evaluate their work themselves whenever they want.

2.2.1 | Architecture

The CLIK judgment system consists of five major components: Yoda instance, Test case, CLIK daemon, CLIK agent, and CLIK repository. We show the overall architecture, components, and procedures of CLIK in Figure 4. We explain each component of CLIK in more detail as follows:

Yoda instance is an instance that runs on the cloud and provides automatic judgments to users (instructor or student). Yoda needs a network address for accessing the automatic judgment service easily and safely from other user’s instances. So, we first set a fixed private IP address for easy access from student’s instances. For example, we used 192.168.0.254, and then, we added a security rule to enable Yoda to access other instances on the network and vice versa. Yoda has a master RSA public key that is paired with the preinstalled RSA private key in each student’s instance. Thus, Yoda can connect, insert the related files, and run the executables on the student’s instance via SSH connection.

Test case is a set of useful and easy scripts for automatic validation of assignments and is created by the instructor; this is the only manual task of the instructor. A test case consists of a test script and a judgment script for an assignment. A test script accepts the input parameters from Yoda and executes the script with the inputs. The outputs are two text files from the kernel and user application. The judgment script takes the outputs and judges whether the test is passed or failed. Thus, the test script and the judgment script are the implementations of the requirements of an assignment. In our experience, they are best created with the Bash shell script [4].

CLIK daemon is a process that always runs on the Yoda instance to wait and respond to requests from instructors or students. CLIK daemon accepts the requests of CLIK users for automatic judgment services, executes
the corresponding test cases by deploying CLIK agents, creates the final report after collecting the results of the CLIK agents, and summarizes the results in the CLIK repository.

CLIK agent is a process that runs on a student’s instance. It is deployed by CLIK daemon and executes the test script. After the tests, the CLIK agent sends the output files to the CLIK daemon. We can execute multiple evaluations of assignments in parallel because Yoda and CLIK agents are physically isolated in the cloud.

CLIK repository is an instance to store the outputs and the scores obtained by comparison with the correct outputs that will be used to produce the report for the instructor. By separating the repository with other instances, we can easily and safely extend the storage without affecting the whole system.

### 2.2.2 Judgment procedure

This section describes the judgment procedure and behaviors of the components in detail. When an instructor evaluates the score of student’s instance with test cases, the CLIK daemon establishes an internal network of the student instances by connecting them to each other and deploys the essential components, such as the CLIK agent, the test script, and any other necessary data, through the network. For example, if the test script requires random inputs, the CLIK daemon produces and deploys the randomized data set to the student instance.

Note that the judgment method is not transferred to the student’s instance because it can be used to cheat. The judgment procedures are executed independently and in parallel so that they can be done in a reasonable time regardless of the number of instances. Thus, CLIK provides a high level of scalability, enough to support many students from several classes.

Next, the CLIK agent automatically builds and executes user applications using the test script. User applications are key parts of OS-course assignments; therefore, students will be attempting to implement them. The role of the applications is to validate the behaviors of the student-made kernel according to the assignment’s requirements. For example, let us assume that an instructor presents an assignment to implement new system calls with specified functionalities. To perform this, a student implements a user application that invokes the new system calls with parameters defined by the assignment. After the execution of the user application, the CLIK agent sends both the outputs of the application and the kernel messages produced by the newly implemented kernel services to the CLIK daemon.

Then, the CLIK daemon compares the correct outputs with the outputs sent by students and judges the result using the judgment method provided by the instructor. The CLIK daemon always knows the correct outputs for test scripts by executing them in advance with the user applications in the instructor’s instance. The judgment method uses the model outputs to judge the outputs from the student instance. We will further explain about the
The judgment method is stored in the CLIK repository and are provided to the student along with their own outputs. The CLIK daemon also delivers a final report that summarizes all the results in the CLIK repository for the instructor. The final reports from CLIK help the instructor to confirm the correct behavior of students’ codes, which is the most time-consuming task during evaluation.

2.2.3 Judgment method

In this section, we further explain the detail of the judgment method. The judgment method cannot be uniform across different assignments because the required behaviors are different. The key consideration is whether the required behavior is deterministic. In our first experience, for example, we demanded the students to implement a simple message queue and new system calls as interfaces for the queue that should be implemented using a First-In-First-Out ring buffer with a bounded capacity. Although the enqueued data are randomly generated, the order of the dequeued data should always be the same. The fullness or emptiness of the queue is also the same when the sequence of enqueue and dequeue requests is the same. If the outputs of the student instances are different from the outputs of the correct version, Yoda reports a fail for the student’s code. Thus, the judgment method is quite simple when the required behavior is deterministic.

On the contrary, our second experience requires a more complicated judgment method. We demanded the students to implement solutions for the readers-writers synchronization problem. To evaluate the student’s answer, the test script runs dozens of threads and the output is the ratio of reading and writing operations that are processed by the kernel. The right solution should avoid the starvation problem. The issue is how to judge the solution successfully by avoiding starvation. In this case, a simple comparison cannot be used to judge the results. We decided on a range of correct answers by using preobtained outputs from the correct version of the kernel and user applications. Due to the different characteristics of various assignments, the judgment method cannot be uniform or automatically generated. Thus, CLIK uses an individual instructor-made judgment method for each assignment. However, the creation of the judgment method is not some arbitrary additional effort required to use CLIK. Some kind of judgment method is essential to any form of evaluation of programming assignments no matter how they are carried out.

3 EXPERIENCE

In this section, we explain our experience with CLIK and provide more details about how CLIK can be used in the real world. We used OpenStack to build the private cloud system for CLIK [15]. The computing infrastructure consists of four high-end servers that are equipped with all-flash storage and connected via a 10 Gbps network.

We presented two kernel programming assignments in a recent OS course in which the number of students participating was over 40, many of them had no experience with cloud systems in which case we had to provide a user manual and a short explanatory session. We also gave them a preliminary assignment on kernel compilation to be sure the students had their own running kernels in the CLIK cloud system before beginning the two primary assignments. The student used a flavor with 12 virtual CPUs, 4 GB main memory, and 40 GB SSD-based storage. The OS image for a student’s instance is based on the Ubuntu 18.04 cloud image but has been modified to change the SSH port number and install the instructor’s RSA public key, as explained in Section 2.1.3. The host name of a student’s instance on the network is configured as that student’s ID for identification of ownership.

3.1 Assignment I: Simple message queue

3.1.1 Requirement

The first assignment is about interprocess communication, it requires the implementation of a simple message queue with two new system calls for sending and receiving. The message queue is implemented as a circular buffer, and the buffer can store a maximum of five items. The send system call enqueues an item, which is an integer number at the head of the queue. The receive system call dequeues an item from the tail of the queue and copies the item to the caller. The kernel returns 0 on success and prints out the predefined message from the enqueued or dequeued item. On failure, the kernel returns −1 and prints out a failure message. The three kinds of failures are defined as follows: “no slot for enqueue,” “no item to dequeue,” and “invalid memory address in the user area.” Two user applications should be implemented to call the send() and receive() functions. A user application requests a send() or receive() function using the syscall() function with input parameters and prints out a predefined message according to the return value. We expect that through this assignment students will obtain knowledge about system calls and the procedure for servicing a system call.
3.1.2 | Test script

We created a test script for the first assignment, pseudocodes of the script are shown in Algorithm 1. The script uses a list for items that store the random numbers transferred from the CLIK daemon. Two text files from the user application and the kernel message are the outputs. Before running the tests, the script clears the queue and kernel messages and, then, loads 10,000 random numbers into a list of items. First, the script enqueues six items to the queue from the list to check the management of overflow. The user application and the kernel should print out the predefined error messages for the overflow on the sixth attempt to enqueue. Next, the six attempts to dequeue will cause underflow and should produce the appropriate error message. After, 994 sets of enqueue and dequeue operations test the robustness of the queue management. Finally, the script enqueues a $-1$ to check the validation code for the input value. After the execution of these tests, we dump the outputs of the user application and kernel into two files. The CLIK agent who executed the script sends the files to the CLIK daemon.

3.1.3 | Results

Figure 5 shows a comprehensive report of the first assignment with 20 students. Although the number of students in our two OS classes in the first semester of 2019 is over 100, we present representative samples in the results for brevity. We have hidden the last digits of student IDs for privacy reasons. We categorize the students who have outputs identical to the correct output as passed. Five students passed and the rest are failed, this was due to several reasons such as no outputs, mistakes in kernel messages, or mistakes in error processing.

We can further investigate the failures of the students using the raw results in the CLIK repository. We ran the CLIK judgment in serial to evaluate the time elapsed for the judgments to be processed. All the judgments took about 184 s to process including the single execution for Yoda, the average time taken was 8.76 s among the 21 executions. Although we additionally check the code and results for a more detailed evaluation, the automatic report from CLIK helps us to quickly validate which the students' code is really working. Note that a student gets the same information as in Figure 5 and that the raw results have remained in the student instance.

This result shows that CLIK can provide automatic evaluation processing for multiple assignments easily. Without CLIK, the instructor would have had to evaluate all 20 assignments individually, there is no doubt that this is a heavy burden.

3.2 | Assignment II: Synchronization

3.2.1 | Requirement

The second assignment is about synchronization and is based on the first assignment. Before the second assignment, we taught the usage of the Linux Kernel Module (LKM) and the Proc File System (ProcFS) to the students.

| Assignment | Student ID | Instance IP | Pass/Fail | Time | Detail |
|------------|------------|-------------|-----------|------|--------|
| simple_queue | 20102391 | 192.168.0.42 | Pass | 12:20:03 10-25-2019 | DIFF: User 0 Kernel 0 |
| simple_queue | 20131090 | 192.168.0.19 | Pass | 12:18:37 10-25-2019 | DIFF: User 0 Kernel 0 |
| simple_queue | 20131314 | 192.168.0.12 | Fail | 12:18:11 10-25-2019 | DIFF: User 2 Kernel 2 |
| simple_queue | 20131573 | 192.168.0.9 | Fail | 12:17:54 10-25-2019 | DIFF: User 996 Kernel 996 |
| simple_queue | 20141421 | 192.168.0.3 | Fail | 12:17:16 10-25-2019 | DIFF: User 2 Kernel 0 |
| simple_queue | 20141426 | 192.168.0.22 | Fail | 12:18:55 10-25-2019 | DIFF: User 2 Kernel 0 |
| simple_queue | 20151467 | 192.168.0.23 | Fail | 12:19:03 10-25-2019 | DIFF: User 2 Kernel 996 |
| simple_queue | 20151468 | 192.168.0.5 | Fail | 12:17:30 10-25-2019 | DIFF: User 2 Kernel 0 |
| simple_queue | 20151474 | 192.168.0.29 | Fail | 12:19:36 10-25-2019 | DIFF: User 996 Kernel 996 |
| simple_queue | 20151475 | 192.168.0.6 | Fail | 12:17:38 10-25-2019 | DIFF: User 996 Kernel 996 |
| simple_queue | 20151476 | 192.168.0.24 | Fail | 12:19:10 10-25-2019 | DIFF: User 0 Kernel 0 |
| simple_queue | 20151706 | 192.168.0.20 | Pass | 12:19:28 10-25-2019 | DIFF: User 0 Kernel 0 |
| simple_queue | 20161719 | 192.168.0.4 | Fail | 12:17:23 10-25-2019 | DIFF: User 2 Kernel 996 |
| simple_queue | 20171105 | 192.168.0.25 | Fail | 12:19:18 10-25-2019 | DIFF: User 2 Kernel 0 |
| simple_queue | 20171111 | 192.168.0.7 | Fail | 12:17:45 10-25-2019 | DIFF: User 0 Kernel 996 |
| simple_queue | 20171636 | 192.168.0.11 | Fail | 12:18:03 10-25-2019 | DIFF: User 2 Kernel 0 |
| simple_queue | 20172106 | 192.168.0.15 | Fail | 12:18:21 10-25-2019 | DIFF: User 2 Kernel 0 |
| simple_queue | 20174646 | 192.168.0.17 | Fail | 12:18:30 10-25-2019 | DIFF: User 2 Kernel 0 |
| simple_queue | 20181215 | 192.168.0.31 | Pass | 12:19:45 10-25-2019 | DIFF: User 0 Kernel 0 |
| simple_queue | 20181842 | 192.168.0.19 | Fail | 12:18:45 10-25-2019 | DIFF: User 2 Kernel 0 |

**FIGURE 5** The summarized report for the instructor with Assignment 1
to help them tackle this assignment. Students were instructed to port their code from the first assignment into LKM using ProcFS instead of system calls.

The assignment requires students to implement the three solutions to the readers–writers problem described in Ref. [16]. The students should prevent a race condition being imposed on the circular queue using appropriate solutions and analyze the solutions with a focus on avoiding starvation. The expected results are as follows. The first and the second solutions may lead to starvation of readers and writers, respectively. The third solution solves the starvation problem, which means readers and writers perform their operations fairly. We provided the usage of semaphore and spinlock services in the Linux kernel for their implementations. Three modules are created as a result of the second assignment, each module implements each of the three solutions.

The students should also implement the ProcFS interfaces for readers and writers. Readers read the most recently enqueued item and writers enqueue or dequeue an item from the circular queue. In addition, common interfaces should be provided to initialize the queue and print out the total number of reads and writes that have been processed since initialization. We provide user applications for readers and writers that use the proposed interfaces. In our test scenario, 18 readers and 18 writers run concurrently for 10 s, this means a total of 36 threads compete for the circular queue.

We expect that students learn about the race condition and synchronization mechanisms such as a semaphore and mutex-lock with a well-implemented code in Linux. In addition, the limitations of the three solutions will teach students the complexity of synchronization problems in real-world computing. The experiences they gain with LKM, ProcFS, and multithreaded programming are also very valuable to them.

### 3.2.2 Test script

The second assignment is hard to evaluate with automatic judgment tools because the behaviors of multithreaded processes are not deterministic. The counters for reading and writing operations will be different for each execution, even with the correct version implemented. To overcome this problem, we carried out a preliminary investigation of the ratio of reading to writing operations for the three solutions. Our preliminary testing gave 7:1. Thus, a ratio below 1:10 or 10:1 will be assumed to give the right answer.

Algorithm 2 shows the pseudo-code for the test script from the second assignment. The script executes the tests that are explained above for the three kernel modules for the three solutions provided by students. At first, the script loads the kernel module to be tested and clears kernel messages. We shorten the test parts for interface and error processing. The messages from the user application are written as a text file. Then, the test with 30 threads running for 10 s is performed. The resulting number of reading and writing operations is printed out by the kernel and also written in a file. The script unloads the kernel module and loads the next module if necessary.

#### 3.2.3 Results

Figure 6 shows the final report of the second assignment for 20 students. The detail section shows the three ratios of the read and write operations for the three solutions. We ran the automatic judgment in parallel, the processing times for each instance are very similar and all within 3 s of each other. As each automatic judgment for the second assignment takes around 30 s, to process judgments in serial would take almost an hour for 100+ students. From this, it's clear to see that this parallel mode is essential for an automatic judgment system of Linux kernel assignments. Three students passed the tests, the rest failed. The failures are not definitive results because the ratios can vary significantly for each execution even if the implementation is correct. For example, the instance with the IP address 192.168.0.9 failed because the second ratio was 1:99, very near to the cut-off of 1:100. This instance passed when we executed the automatic judgment again. Other students failed for various reasons such as their code not working, one or two solutions not providing results, and incorrect results. These students will modify their codes and check their results again using the CLIK agent in their instance without the involvement of the instructor.

### 3.3 Evaluation of parallel judgment

We conduct experiments to evaluate the parallel judgment by measuring execution time when running test scripts. Thus, we design two experimental cases: baseline and CLIK. The baseline executes all test scripts in serial, and CLIK executes them in parallel. The behaviors of test scripts are described in Section 3.1.2 (Assignment 1) and 3.2.2 (Assignment 2). The results are shown in Figures 7.
and 8 as bar graphs. In the graphs, the dark bars and the light bar represent the results of Assignment 1 and Assignment 2, respectively.

Figure 7 shows the total elapsed time for evaluating correct test scripts. In the graph, the x-axis indicates comparison targets, the baseline (left two bars) and CLIK (right two bars), and the y-axis indicates the elapsed time (seconds). The baseline evaluates test scripts sequentially, and the number of correct test scripts is 20 in Assignment 1 and 17 in Assignment 2, respectively. In the graph of baseline, we observe that the test scripts of Assignment 1 have shorter execution time than ones of Assignment 2. For convenience, we call each as short and long scripts.

For CLIK, we also applied the same test scripts and measured the total elapsed time. From the results of Figure 7, we infer that CLIK reduces the elapsed time by up to 15 times or more in long scripts and 18 times or more in short scripts.

To analyze the reason of our enhancement, we also individually measure the elapsed time of executing correct test scripts, and Figure 8 shows the results. In the graph, the x-axis indicates the number of student instances, and the y-axis indicates the elapsed time (seconds). In the graph, we observe that 20 and 17 students are passed the requirements of Assignment 1 and Assignment 2, respectively. The elapsed times of Assignment 1 are distributed from 5.64 seconds (min.) to 6.69 seconds (max.), and the average time is 6.25 seconds. The elapsed times of Assignment 2 are distributed from 35.27 seconds (min.) to 42.56 seconds (max.), and the average time is 38.06 seconds. The difference in execution time can vary depending on the result of CPU scheduling. In the experimental environment, 20 instances share with 132 physical CPUs on three physical servers. The default scheduling time quantum in Linux is 20 ms, and the typical number of processes is over 100, even in the idle server. Thus, the delay time of 10–20 ms occurring in CPU scheduling may cause an increase in the execution time of 1–2 seconds or more. Also, the time to send the results to the Yoda instance can make a slowdown if the network device is busy or the CLIK daemon process does not take a scheduling opportunity.

Without CLIK, we have to execute the scripts one by one, and the total execution time will be the sum of each elapsed time. However, with CLIK, we can execute all test scripts in parallel; thus, the total execution time is determined by the longest execution time in the executed test scripts (6.69 s in Assignment 1 and 42.56 s in Assignment 2).
Therefore, from Figures 7 and 8, we can ascertain that CLIK is also beneficial for evaluating a lot of assignments in terms of elapsed time.

3.4 | Resource usages

This section provides the usages for pCPUs and memories that are collected for our two 2019 OS courses. We collect the CPU utilization and memory usages from two of three computing nodes and the controller node using the Sar monitoring tool. Unfortunately, we failed to collect the data from one of the computing nodes due to a misconfiguration. The missed one has the lowest computing power and memory, 28 pCPUs and 64 GB memory. The two other nodes have 104 pCPUs and 512 GB memory in total. The controller node has 40 pCPUs and 64 GB memory.

Figure 9a illustrates the CPU usages of three nodes for about three months. The number is average for an hour and is collected from both user and kernel. At first, we provide the preliminary assignment for the students to learn the usage of the CLIK cloud system and kernel compilation before April. The first assignment is announced on the first May after the reboot due to the system error. The deadline was May 22 and the day shows the highest CPU utilization of 67.36% for computing node 1. The second assignment is announced right after and the deadline was June 17. On average, the CPU usage for the second assignment is lower than the first assignment because the second one does not require the many kernel compilation compared to the first one which requires creating new system calls. The CPU usage of the controller node is stable and low for the whole period.

Figure 9b shows the memory usage of three nodes, and to depict the memory size in the graph, we measured the on-average amount of memory size for an hour. The graphs show similar trends to Figure 9a before the deadline for the first assignment. The compute node 1 was reached to its maximum physical memory, 256 GB, around the first assignment’s deadline. CPUs released are immediately returned to OS; however, it takes some time to release the allocated memory to OS. Thus, the highly utilized memory situation of the compute node 1 is continued for a while despite passing the deadline. Then, the memory usage becomes more relieved for the period of the second assignment. However, the size of memory usage of computing node 1 was still higher than that of node 2, and even it sometimes reaches the maximum capacity. The controller node almost always consumed the memory as much as close to its maximum memory capacity because the instance images were always buffered in the memory.

The implications from the resource usages are following. At first, the computing power and memory capacity of our physical servers are sufficient to serve student instances of 100 or more required for the Linux kernel-based OS assignments. Some students who submitted the first assignments at the deadline were troubled by the slowdown of student instances; however, other students who submitted their assignments before the deadline were not. If the OpenStack instance placement among the computing nodes is more balanced, we believe
that the efficiency of resource usages will be enhanced. Second, we found that memory is a more valuable resource than CPU for evaluating assignments because the memory for instances is usually allocated in a huge size of 4 GB but released lazily. Thus, a computing node with more memory will provide a more pleasant environment for the students and the memory capacity should be carefully decided when we prepare the flavor.

We did not provide the usage of the storage device because there were no valuable findings. The student instance almost uses its 40 GB storage capacity after the kernel compilation is performed once. Thus, almost 4 TB of storage capacity is used for serving 100 student instances.

3.5  |  Lessons

Our experience with CLIK taught us several lessons. The main lesson is that CLIK makes the management of programming assignments with OS kernels much easier. Without the CLIK cloud, students would have had to use their machines to create the appropriate environment for the assignments. We have observed in the past that this

![Figure 9](image-url)
causes difficulties to not only the students but also the instructors because access to the code of students who needed help was very inconvenient. For example, some students would visit the office of an instructor with their laptops. Others who use their personal computers at home would enable remote access for their environments. These different environments and access methods waste the instructor’s time.

The CLIK cloud solves this problem by providing consistent and convenient environments and access methods for students and instructors. In this process, we observed that some students asked questions about CLIK cloud usage. Thus, we provided a clear and simple user manual that describes exactly how to use the CLIK cloud to avoid any misunderstandings and make using the CLIK cloud even more straightforward. Also, to avoid wasting time while adapting to new environments and their associated requirements, the CLIK cloud provides a consistent environment. With the CLIK cloud, the students who need help for their assignment need only send an e-mail with their private IP on CLIK to the instructor without needing to visit his/her office. Furthermore, the automatic judgment of CLIK helps students to check their progress in the current subject in a timely fashion. We also observed that instructors could reduce the time they spent evaluating assignments with automatic judgment because CLIK had already prepared the evaluation environment and tested the assignments. Thus, instructors do not need to spend their time building kernel code and rebooting the system for testing. Moreover, the automatic judgment generates comprehensive reports for all students only with a single command on an instance of Yoda. Thus, we observed that automatic judgment also minimizes the effort needed to manage and assess assignments.

Other miscellaneous lessons are as follows: (a) providing skeleton codes with preformatted output text is a good idea if it supports the purpose of the assignment. (b) The requirements for computing resources should be thought about carefully when deciding the instance specification. For example, we can calculate the number of vCPUs, memory capacity, and storage capacity according to our physical resources. However, the instance snapshot makes precalculation difficult. If students try to back up their code by using full snapshotting of their instances, the size of the snapshots can consume a huge storage capacity of 40 GB. If we reserve the required computing resource considering a full snapshot capacity, we can only support a few students. To mitigate this capacity issue, we provide guidance for resource allocation, allowing only one snapshot for each student. (c) An effective way needs to be developed beyond warnings about performance degradation issues near deadlines as we have observed that the performance of each instance slows down when the assignment deadline approaches. Thus, we warn the students about the performance issue of the cloud system from the beginning of the assignment. However, many students have still suffered from this performance issue around the deadline. (d) The web-based graphical user interface (GUI) for CLIK is not effective. As per our experience, the command-line interface (CLI) was enough to provide the necessary functionalities. The CLI is convenient because the environments for the assignment and judgment are the same. Thus, to avoid unnecessary effort we decided to not create a GUI.

4 | RELATED WORK

In terms of utilizing core technologies, we categorize previous studies of interest into three classes: web-based, virtualization-based, and cloud-based approaches. We summarize our analysis of related literature in Table 2.

First, two studies, [8] and [1], involve web-based approaches. Commonly, web-based studies focus on easy accessibility and interactivity. WebUbu [8] suggests a web-based environment for interactive OS simulation and evaluation. WebUbu can perform various Linux command processing but more complicated system programming assignments such as direct modification of the Linux kernel are hard to achieve. Another study, WebExam [1], provides a web-based automation tool for the evaluation of online examinations; this is very useful for instructors but it has the same limitation as WebUbu, that is, it also does not support real-world system programming evaluation.

Second, looking at the virtualization approach, four studies, [7,11,12,14], have been published. The virtualization technique provides the illusion that a single user has a single computer, even though there are multiple users, by isolating and demultiplexing physical hardware at the system level. Nieh and Vaill [14] suggest a virtual kernel development environment using virtual machines. They showed that a virtual kernel development environment could be successfully deployed and utilized for kernel-level assignments with students. Another study [11] shows that a virtual kernel-level assignment environment can even support large classes by using a distributed version control system such as Git. Another study, VirtEmbed [12], also makes a cost-effective lab environment for embedded system education by virtualization. Additionally, VirtSAN [7] applies virtualization for storage area network (SAN) education, and they also
## Table 2: Comparisons with related literature

| Study (year) | Environment | Benefits for instructors                        | Benefits for students                                      | Kernel programming support | Scalable resource control |
|--------------|-------------|------------------------------------------------|------------------------------------------------------------|-----------------------------|---------------------------|
| WebUbu (2016) | Web         | Easy tracking of students’ performance         | Easy-to-use Linux commands                                 | X                           | X                         |
| WebExam (2019) | Web         | Easy evaluation via online exam                | Easy accessibility and interactivity                        | X                           | X                         |
| VirtLinux (2006) | Virtualization | Reducing cost of administering kernel assignment | Easy deployment of kernel-level assignment                   | X                           | X                         |
| VirtAppliance (2010) | Virtualization | Reducing cost of administering kernel assignment + tracking version control | Easy deployment of kernel-level assignment + distributed version control | O                           | X                         |
| VirtEmbed (2012) | Virtualization | Reducing cost of administering embedded assignment | Cost-effective deployment of embedded kernel-level assignment | O                           | X                         |
| VirtSAN (2015) | Virtualization | Reducing cost of administering SAN assignment  | Cost-effective deployment of SAN environment                | X                           | X                         |
| ELCloud (2015) | Cloud       | Scalable and Integrated knowledge collection    | Easy-to-use virtual testbed                                 | X                           | O                         |
| Edu-Cloud (2018) | Cloud       | Easy-to-manage virtual testbed                 | Easy-to-use colearning platform and virtual testbed         | X                           | O                         |
| CloudLab (2012) | Cloud       | Easy-to-manage virtual testbed                 | Easy-to-use virtual network testbed                        | X                           | O                         |

Abbreviation: SAN, storage area network.
mention that virtualization can reduce the cost and effort that is required for building real-world SAN facilities.

Third, we reviewed three studies that took a cloud-based approach. ELCloud [17] suggests a knowledge-based e-learning system in cloud computing. The authors focused on content delivery power related to e-learning applications in terms of network bandwidth, storage, processor flops, and physical memory. However, they did not cover the OS-course-specific factors such as kernel programming development. Edu-Cloud [18] suggests another cloud-based e-learning system and provides some use cases such as a social colearning platform and virtual testbeds. The authors focused on resource provisioning and designed a resource selection algorithm based on available resources. However, the study did not cover practical programming assignments or evaluations. Another study [6] proposes a cloud-based solution for teaching about computer networks. In Dinita et al. [6], the authors built a network testbed with cloud computing and showed the various benefits of a virtual infrastructure such as ease of use and flexibility.

The common drawback of these studies is that they lack the evaluation of practical programming assignments elements for tasks like kernel programming that is essential in the OS course.

5 | CONCLUSION

In summary, this paper presented CLIK, a cloud-based Linux kernel practice environment that includes an automatic judgment system. We claimed that this cloud system satisfies the important requirements for a practice environment using the Linux kernel in terms of a unification of the execution environment that maintains isolation between students. The CLIK cloud is our first deployment of the OpenStack-based private cloud system. We provide detailed configurations and descriptions of the CLIK cloud including information about the virtual network, flavor, and image. Next, we introduced CLIK judgment, an automatic and bidirectional judgment system deployed on the CLIK cloud. Our experience with two OS assignments showed that CLIK judgment works well in the real world and that the effort required to verify a student's work is minimized with its use. In addition, we discussed several key takeaways from using CLIK that will be helpful to design and operate similar judgment systems for OS assignments. Finally, we provided the lessons learned in designing, implementing and operating CLIK that could be invaluable while developing similar automatic judgment systems for practical OS assignments.

### Algorithm 1. Test Script for Assignment 1

| Data: List for items |
|---------------------|
| Input: Random numbers from CLIK daemon |
| Output: Two text files |
| 1 Dequeue five items to empty the queue; |
| 2 Clear kernel messages; |
| 3 Load 1,000 random numbers into a list of items; |
| for 6 times do |
| 5 Pop an item from the list; |
| 6 Enqueue the item; |
| 7 for 6 times do |
| 8 Dequeue an item; |
| for 994 times do |
| 14 Dump user application messages to a file; |
| 15 Dump kernel messages to a file; |

### Algorithm 2. Test Script for Assignment 2

| Input: Three kernel modules for three solutions |
|---------------------|
| Output: Two text files for each kernel module |
| 1 for three kernel modules do |
| 2 Load the kernel module for the next test; |
| 3 Clear kernel messages; |
| 4 Test the interface and error processing with a single-threaded user application; |
| 5 Dump user application messages to a file; |
| 6 while 10 seconds do |
| 7 Run the 15 writer and 15 reader threads; |
| 8 Dump the numbers of read and write operations in kernel message to a file; |
| 9 Unload the kernel module; |

### ACKNOWLEDGMENTS

This study was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT; No. NRF-2020R1F1A1067365, NRF-2017R1C1B5016000, and NRF-2020R1G1A1005544) and in part by the Research Base Construction Fund Support Program through the Jeonbuk National University in 2020. The authors would like to thank the anonymous reviewers of Computer Applications in Engineering Education Journal for their valuable comments and suggestions to improve the quality of the paper. The authors
also thank Myeonggyu Lee, who helps the implementation of CLIK very much.

**ORCID**

Hyunchan Park [ORCID](http://orcid.org/0000-0001-9879-5531)

Youngpil Kim [ORCID](http://orcid.org/0000-0002-9154-2336)

**REFERENCES**

1. F. Al-Hawari et al., *Integrated and secure web-based examination management system*, Comput. Appl. Eng. Educ. 27 (2019), no. 4, 994–1014.

2. Apache CloudStack. [https://cloudstack.apache.org](https://cloudstack.apache.org). Accessed January 14, 2020.

3. AWS Cloud Computing for Education. [https://aws.amazon.com/education](https://aws.amazon.com/education). Accessed January 14, 2020.

4. Bash shell script. [https://www.gnu.org/software/bash](https://www.gnu.org/software/bash). Accessed January 14, 2020.

5. M. Bhatia, S. K. Sood and S. Kaur, *Quantum-based predictive fog scheduler for IoT applications*, Comput. Ind. 111 (2019), 51–67.

6. R. I. Dinita et al., *A cloud-based virtual computing laboratory for teaching computer networks*, 13th International Conference on Optimization of Electrical and Electronic Equipment (OPTIM) IEE, Brasov, 2012, pp. 1314-1318.

7. J. García and J. Entrialgo, *Using computer virtualization and software tools to implement a low-cost laboratory for the teaching of storage area networks*, Comput. Appl. Eng. Educ. 23 (2015), no. 5, 715–723.

8. A. Garmpis and N. Gouvatos, *Design and development of WebUbu: An innovating web-based instruction tool for Linux OS courses*, Comput. Appl. Eng. Educ. 24 (2016), no. 2, 313–319.

9. Google Education Programs. [https://edu.google.com/programs](https://edu.google.com/programs). Accessed January 14, 2020.

10. H. Kaur, S. K. Sood and M. Bhatia, *Cloud-assisted green IoT-enabled comprehensive framework for wildfire monitoring*, Cluster Comput. (2019), 1–14.

11. O. Laadan, J Nieh and N. Viennot *Teaching operating systems using virtual appliances and distributed version control*, Proceedings of the 41st ACM Technical Symposium on Computer Science Education ACM, Milwaukee, WI, 2010, pp. 480-484.

12. H. Lim, H. Yu and T. Suh, *Using virtual platform in embedded system education*, Comput. Appl. Eng. Educ. 20 (2012), no. 2, 346–355.

13. A. Manocha, R. Singh and M. Bhatia, *Cognitive intelligence assisted fog-cloud architecture for generalized anxiety disorder (GAD) prediction*, J. Med. Syst. 44 (2020), no. 1, 7.

14. J. Nieh and C. Vaill, *Experiences teaching operating systems using virtual platforms and Linux*, ACM SIGOPS Oper. Syst. Rev. 40 (2006), no. 2, 100–104.

15. OpenStack. [https://www.openstack.org](https://www.openstack.org). Accessed January 14, 2020.

16. Readers–writers problem from Wikipedia. [https://en.wikipedia.org/wiki/Readers-writers_problem](https://en.wikipedia.org/wiki/Readers-writers_problem). Accessed January 14, 2020.

17. R. Sandhu and S. K. Sood, *A commercial, benefit driven and secure framework for e-learning in cloud computing*, Comput. Appl. Eng. Educ. 23 (2015), no. 4, 499–513.

18. S. Shahzadi, M. Iqbal and T. Dagituklas, *Edu-Cloud: On-the-fly employability skills as a service*, Comput. Appl. Eng. Educ. 26 (2018), no. 2, 314–325.

**AUTHOR BIOGRAPHIES**

Hyunchan Park received the BS, MS, and PhD degrees in computer science from Korea University, Seoul, South Korea. He was a Research Professor with Korea University from 2014 to 2016. He is currently an Assistant Professor with the Division of Computer Science and Engineering, Jeonbuk National University. His current research interests include operating systems especially with flash-based storage devices, virtualization systems, and cloud computing.

Youngpil Kim received the BS, MS, and PhD degrees in computer science and engineering from Korea University, Seoul, South Korea, in 2002, 2004, and 2015, respectively. After completion of his degree, he was appointed as a research professor in the College of Informatics at Korea University. In 2019, he joined the School of Computer Science, Semyung University, Jecheon, South Korea, where he is currently an assistant professor. His research interests include system reliability, operating systems kernel architecture, cloud computing, and system-level security.

**SUPPORTING INFORMATION**

Additional supporting information may be found online in the Supporting Information section.

**How to cite this article:** Park H, Kim Y. CLIK: Cloud-based Linux kernel practice environment and judgment system. *Comput Appl Eng Educ.* 2020;28:1137–1153. [https://doi.org/10.1002/cae.22289](https://doi.org/10.1002/cae.22289)