VR City: A Tool for Understanding Control Flow in Linux Kernel

Shuanghui Yi[1], Xiaoyan Su[1], Tong Wang[1], Shaobo Yu[1], Bingru Wu[2,*], and Hang Li[2]

[1]Systems Engineering Institute, Academy of Military Sciences, Beijing, China
[2]School of Computer Science and Technology, Software Engineering, Xidian University, Xi'an, China
Email: yshbise@foxmail.com (S.Y.); suyan_sxysxy@foxmail.com (X.S.); tongwss@foxmail.com (T.W.); qilinbo@foxmail.com (S.Y.); 21151213685@stu.xidian.edu.cn (B.W.); hangli@mail.xidian.edu.cn (H.L.)
*Corresponding author

Manuscript received July 3, 2024; revised August 16, 2024; accepted August 22, 2024; published February 25, 2025

***Abstract*—The current code size of the Linux kernel has exceeded 1 GB, with an extremely complex control flow. This complexity poses at least two obstacles for students to correctly understand the causal paths in the code: 1) The tendency to get lost; 2) Lack of concretization, making it difficult for learners to use constructivist approach to understand the Linux kernel. To address these obstacles, this study leverages VR (Virtual Reality) technology to concretize the Linux kernel as a city. For example, it represents Linux kernel function codes as buildings in the city and the control flow as running vehicles. This approach aids learners in comprehending the overall code structure, establishing a macro-level map of the code. By concretizing the concepts, learners can relate their existing rich knowledge of daily urban transportation to Linux kernel knowledge. This method facilitates the application of constructivist learning, reducing the difficulty of understanding the Linux kernel.**

***Keywords*—VR city, Linux kernel, visualization**

I. Introduction

With the updates in Linux kernel versions, the codebase has now exceeded 1 GB. The Linux kernel itself exhibits high concurrency and incorporates a significant number of function pointers, leading to an exponential growth in the potential number of control flows [1]. Consequently, learners face immense difficulty comprehending the kernel, often getting lost in the complexities. Moreover, the code, being an abstract collection of two-dimensional characters, poses challenges in bridging the gap between code semantics and the learners' existing knowledge, thus hindering the application of effective constructivist methods and increasing the cost of understanding Linux kernel code.

To address the aforementioned issues, this paper employs virtual reality technology to visualize the Linux kernel as a city, called VR city. It maps folders and files within the Linux kernel codebase into nested regions within the city. Functions in the files are represented as buildings within these regions, and the control flow is depicted as moving vehicles. This approach assists learners in comprehending the overall code structure and creating a macro-level map of the code. By employing such visualization, learners can connect their knowledge of the Linux kernel with their extensive everyday knowledge of urban traffic, facilitating the use of constructivist methods to comprehend the Linux kernel, thereby reducing the difficulty in understanding it.

In addition, VR has stronger immersion than normal 3D virtualization. Normal 3D virtualization transmits information through a two-dimensional display plane. In this case, the human eye not only receives the useful information output by the display, but also receives the useless information output by the surrounding environment, which causes attention to drift. After wearing a VR device, the human eye's field of view is completely covered by the VR device. The human eye can only obtain information provided by the VR device, and cannot obtain other information, which increases immersion. High immersion is conducive to students to eliminate distractions and learn knowledge. This is why VR is chosen, not normal 3D virtualization.

II. Related Work

Virtual Reality (VR) technology, representing a new generation of information technology, has been widely applied in the field of education. The core reason behind this lies in the 3I characteristics of VR, namely: Imagination, Interaction, and Immersion [2]. These features bring several benefits to learners:

1) The imaginative aspect of VR combines the rational and emotional aspects of learners, allowing the creation of artificially imagined scenarios or objects, deepening concepts, and fostering new ideas;
2) VR's interactivity enables learners to receive feedback through multiple senses, providing an immersive and interactive learning experience, without relying on traditional computer devices like keyboards or mice. This allows for a more natural interaction with the virtual environment, thus reinforcing the impression of acquired knowledge;
3) The immersive nature of VR facilitates the transformation of learners from observers to active participants, allowing them to feel present in the virtual world. This shortens the distance between learners and tedious code, ultimately increasing engagement and initiative in the learning process [3].

For instance, in 2019, Krokos' study indicated that students retained more information and applied their acquired knowledge better after engaging in VR exercises. In 2020, Alalwan conducted research combining virtual reality, augmented reality, and mixed reality to explore their applications in education.

Moreover, to visually represent code complexity, a software analysis integrated environment named "Code City" [4] was developed. In this environment, the software system is visualized as an interactive and navigable 3D city, where classes are depicted as buildings within the city, and packages are described as regions where these buildings are located.

In Code City, code has already been visualized as a city.

However, we have combined VR technology with 3D models to develop a new technology called VR City. Compared to Code City, VR City has the following key features:

1) The goal of VR City is to help students understand the Linux kernel;
2) VR City integrates the teacher's prior knowledge with static 3D models of the code, effectively presenting the core architecture of highly complex software;
3) VR City can dynamically display program control flow, reducing the cost of understanding complex control flow;
4) VR City is presented through VR devices, providing users with better interactive experience and better immersion.

In summary, our paper visualizes a city embodying the program control flow of the Linux kernel code in VR devices. The movement of cars within buildings symbolizes processes traversing between different parts of the codebase. Furthermore, to the best of our knowledge, we have not found any prior researchers have conducted such studies.

III. Concrete Design And Implementation

Concretization is a technique that transforms abstract concepts into tangible representations [5]. In this paper, the design of concretization adheres to the following three principles:

Principle 1: The concretization corresponding to the code should be familiar and understandable to the learners.

This principle ensures that the cost for learners to comprehend the concretization is minimized, preventing an increase in the learning curve. Following this principle, this paper represents the Linux kernel source code as a city and depicts the control flow as vehicles moving within this city.

Principle 2: The cost of building concretization should be significantly lower than that of constructing the Linux kernel source code.

This principle ensures the efficiency and usability of the concretization model. Otherwise, the concretization model would become overly complex, possibly surpassing or equaling the complexity of understanding the Linux kernel source code, rendering the model less valuable. Following this principle, this paper focuses on concretizing key portions of the Linux kernel source code based on the learner's objectives.

Principle 3: Users should have a clear understanding of their current position in the control flow at any given time.

This principle aims to reduce the probability of users getting lost in the concretization model. Accordingly, this paper adopts a hierarchical representation to illustrate the user's control flow during concretization.

A. Design Principles

Before conducting the concretization design, it's essential to comprehend the structure and functionalities of the object to be visualized. Here are the structure and functions of the Linux kernel [6]:

The Linux kernel source code comprises numerous folders, their corresponding subfolders, and source code files. These source code files contain function code. When running a Linux system, processes are created and these processes will call functions in the source code.

Having understood the structure and functionalities of the Linux kernel, we will attempt to concretize it. The process of concretization is as follows [7]:

The Linux kernel, as a whole, is concretized as a city; in the Linux kernel source code, folders and files are concretized as multiple districts within the city; in the Linux kernel source code, folders often contain many subfolders and source code files, and these subfolders and source code files are concretized as subdivisions of districts, and even further as sub-districts; for the functions in source code files are concretized as buildings within the districts; the static call relationships between different functions in the Linux kernel source code are concretized as roads within the buildings [8]. The overall mapping relationship of the software is illustrated in Fig. 1.

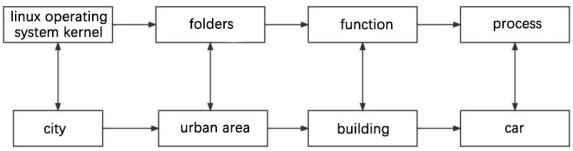

Fig. 1. The overall mapping relationship of the software.

Through the aforementioned mapping relationships, the abstract Linux kernel code can be visualized as a city, presenting complex and abstract concepts of the operating system in a more vivid and intuitive manner. Next, let's illustrate the application value of this mapping relationship by focusing on the visualization of function call stack.

Within the Linux kernel, the intricate function call stacks have always been a challenging aspect of the visualization model. To vividly display the call stacks, this software uses a planar representation. Specifically, all functions within the Linux kernel correspond to buildings on the first level plane. When a process calls a function, the buildings corresponding to the called function and all its child functions are placed on the second level plane. This sequence continues, creating a new plane for each subsequent function call. As the call stack grows, a new plane is established on the existing plane, including the buildings corresponding to the called function and its child functions. The call stack mapping relationship is illustrated in Fig. 2.

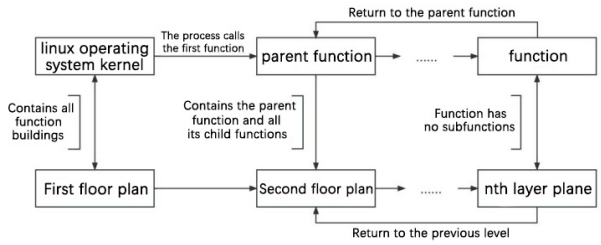

Fig. 2. Call stack mapping relationship.

B. Static Scene Design Details

We will elaborate on the various elements of the concrete model. Initially, we'll delve into the different aspects of static concretization, from the city down to districts, and then onto buildings.

1）City

We employ a city to represent a Linux kernel. At present we build five districts in the city: Process Scheduling (SCHED), Memory Management (MM), Virtual File System (VFS), Networking (NET), and Inter-Process

Communication (IPC). Their source code is stored in respective folders within the kernel source code files - kernel, mm, fs, net, and ipc folders. The setup of the district model is illustrated in Fig. 3.

2） Buildings

Buildings are used to represent functions within the source code files. Districts consist of numerous buildings, with each building within a district representing functions included in the files of that district. These buildings represent actual functions that need to be executed in Linux kernel. Each building's attributes mirror the respective function's attributes. For instance, the building's name signifies the function's name, and its description represents the function's purpose. To ensure a clearer and more intuitive model, the design of buildings should align with the characteristics of the functions, thereby visually displaying various functions' features. For example, a building that represents a function like "do_task_dead", responsible for destroying cars, can be styled like a factory. Furthermore, different functions possess varying levels of importance, and consequently, their corresponding buildings should reflect these differences. Specifically, functions can be categorized into three levels of importance. Buildings representing the most crucial functions could be skyscrapers, those representing moderately important functions could be modeled high-rise residential buildings, and those representing the least important functions could be modeled bungalows.

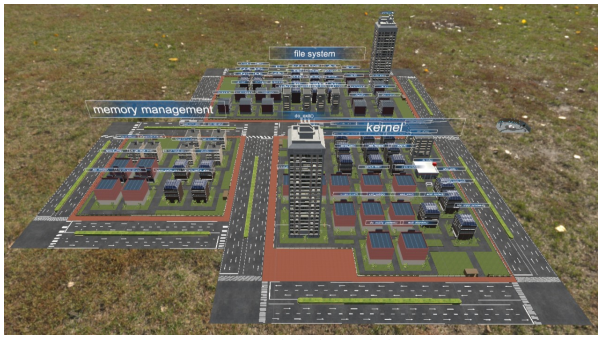

Fig. 3. Aerial view of city.

C. Dynamic Execution Design Details

Now, we introduce dynamic visualization, specifically the moving cars.

1） Process cars

In the program control flow visualization software, we will represent running processes with cars, called process cars. The appearance of a process car is similar to that of a common sedan, characterized by a progress bar indicating the progress status. The modeling of process cars is depicted in Fig. 4.

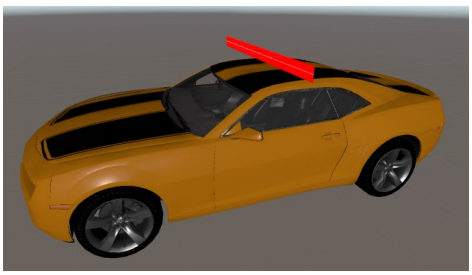

Fig. 4. Process car.

Navigation system. The car's automatic path-finding module is implemented using Unity automatic navigation components. The baking result of the navigation system is shown in Fig. 5.

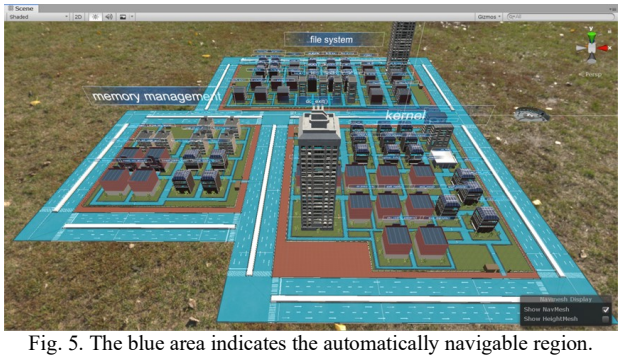

Fig. 5. The blue area indicates the automatically navigable region.

2） UI interface

Menu bar. Players interact with the software through the menu interface to control perspective switching, change the car's driving speed and other operations. The menu bar is shown in Fig. 6.

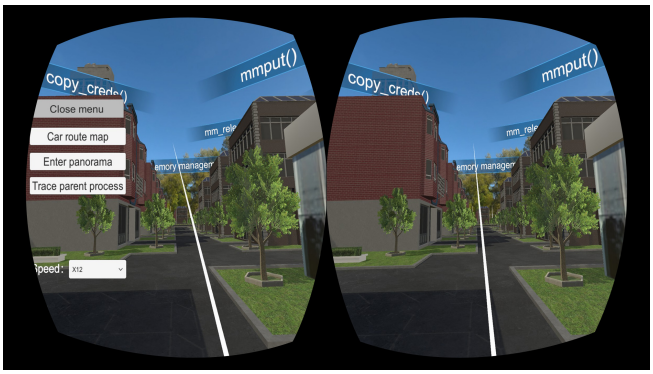

Fig. 6. Menu bar.

Car information window. When the controller clicks the car route map button or enters the car perspective, a translucent car information window which includes process-related information pops up on the right side. In this window, click the details button in the task list window being executed to pop up all the program control flows that the process needs to execute. As shown in Fig. 7.

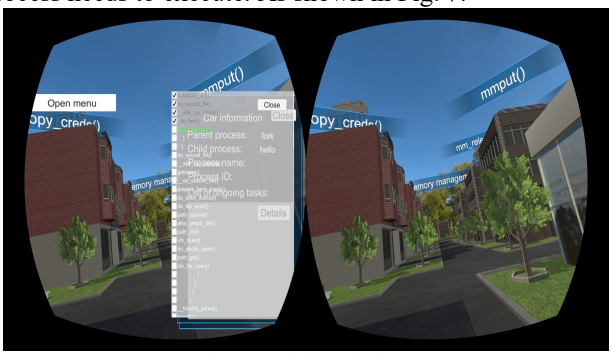

Fig. 7. Task list window.

Building information window. Click on the building from the scene perspective to pop up the building information window. The building information window displays the name and function introduction of the function represented by the building, and includes the function call list which displays all sub-functions of the function. The building

information window is shown in Fig. 8.

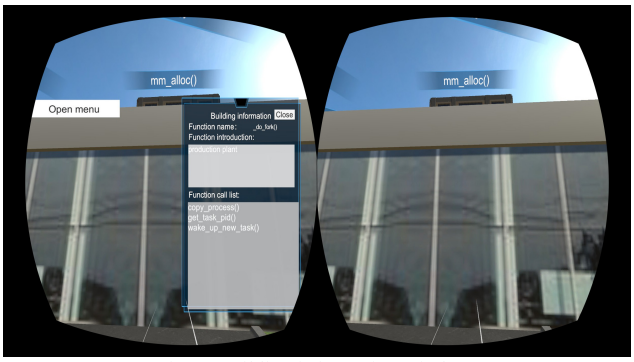

Fig. 8. Building information window.

Speed. There is a speed drop-down list in the menu bar. Selecting any one of the options can change the driving speed of all cars in the scene. The vehicle speed is determined by the speed attribute whose unit is meters per second in the automatic pathfinding component Navigation Mesh (NavMesh) agent. For example, the X6 has a vehicle speed of 6 meters per second. The speed dropdown list is displayed as shown in Fig. 9.

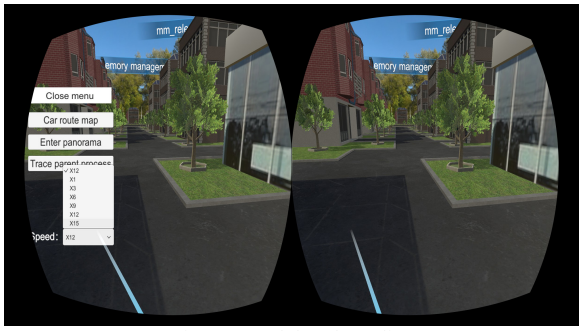

Fig. 9. Speed dropdown list.

3） Perspective

This software has three perspectives:

1) Free perspective. It will allow learners to freely explore the scene through the first-person perspective, avoiding the negative impact of boring and abstract code on learners, thereby enhancing their interest in learning and deepening their impression and understanding of the code;
2) Car perspective. Learners can ride in the process car and experience the control flow journey of the kernel, transforming learners from observers to participants, and shortening the distance between learners and kernel;
3) Panoramic perspective. It is for the camera to overlook the city, making it convenient for students to observe the entire VR City and the running routes of cars, thereby quickly establishing a clear macro-understanding of the Linux kernel.

In free perspective, user can do following operations:

1) Click the ground with the controller to move to the clicked position;
2) Turn the head sensor to rotate the perspective;
3) Click on the building to pop up the building information window.

When the controller clicks the panorama mode button in the menu, the perspective will jump to the panoramic scene. The panoramic view is shown in Fig. 10.

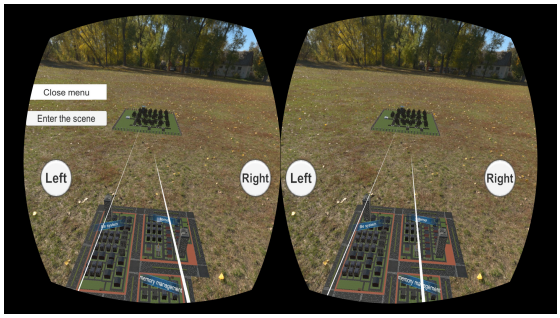

Fig. 10. Panoramic perspective.

Now we click “Enter the scene” button to return to free perspective. In free perspective, by clicking the track parent process button or track child process button with the controller, the user can switch to the car perspective. The “Track parent process” and “Track child process” buttons in the drop-down menu are hidden, the “Get off” button is displayed, and the car information window pops up. The car perspective is depicted in Fig. 11.

Fig. 11. Track parent and child processes from car perspective.

4） Layering

When VR city is running, on the main plane, let the plane be P1, the car drives to the first building, and the function represented by the building is B1, which means that the process calls the first function. If B1 needs to call a function, then in order to clearly represent the function call stack, we need to establish a secondary plane, let this plane be P2. On P2, it includes B1 and the buildings corresponding to all functions called by B1. The set of all functions called by B1 is set to B1S1. If a function in B1S1, set it to B2, needs to call other functions, then you can create a new plane in the same way and set it to P3. P3 contains B2 and the buildings corresponding to all functions called by B2. The hierarchy is depicted in Fig. 12.

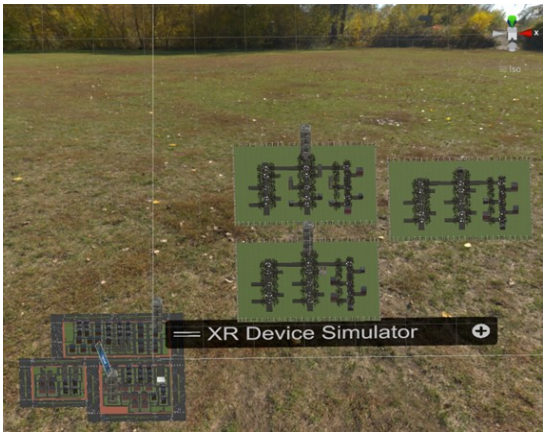

Fig. 12. Stepwise hierarchy.

IV. Experimental Results

In order to verify the effectiveness of VR-based program control flow visualization research, we carried out activities to use VR City to teach selected students some subsystems of the Linux kernel, and distributed questionnaires to them.

To ensure the feasibility and validity of the experiment, we selected 60 students from the operating system course design experiment course at XiDian University as subjects. All students were between the ages of 18 and 22, with similar intelligence levels.

The experimental design is as follows:

First, all students are given the same test, called the first test, and scored. The first test is designed to assess students' basic understanding of Linux system calls.

Then, the subjects are evenly randomly divided into two groups A and B, with 30 people in each group. The division results should ensure that the average scores and variances of the two groups of students in the first test are close, otherwise the division should be redone.

Next, different teaching methods are used to teach the two groups of students. An experienced teacher records a video lecture in advance. Group A learns by watching the video lecture. Group B learns by using VR City and listening to the audio portion of the lecture. It is important to note that the audio portions of the lecture that the two groups listen to are identical and include the internal control flow of Linux system calls. This is done to avoid the influence of teachers' bias on the experimental results. The lecture time for both groups is 60 minutes.

Finally, both groups of students are re-tested again, called the second test, and scored. The second test is designed to assess students' understanding of the internal control flow of Linux system calls.

A. Select Experimental Subjects

Screen students who meet the following requirements in advance through questionnaires:

1) Have a basic understanding of the Linux kernel and have done Linux system call programming;
2) Interested in the internal control flow of Linux system calls.

The reasons for screening these students are twofold. First, learning should be incremental, from outside to inside, and from shallow to deep. Students who are completely unfamiliar with the Linux kernel will not be able to quickly understand the control flow with the help of VR City, and they may also become confused by the various basic concepts of Linux. Second, this experiment is designed for students who are interested in understanding the internal control flow of Linux system calls. Therefore, students who are not interested in this topic will not be considered.

Based on the above criteria, we decided to select participants from the operating system course design experiment course at XiDian University. The reasons are as follows:

On the one hand, students who can choose this course are all sophomore computer science students. They have already learned relevant knowledge of operating systems and have a certain understanding of operating systems. On the other hand, the operating system course design experiment is an advanced course. Learning the internal control flow of Linux system calls helps to complete the course, so students are more interested in participating in the experiment [9,10].

In summary, these students meet the requirements.

B. Experimental Process

The experimental process is listed as follows:

On the day of the experiment, a pre-test was conducted from 9:00 to 10:00 AM. The subjects were randomly divided into two groups, A and B. If the average scores and variances of the two groups in the pre-test are too different, then the groups must be re-divided. This is because if the groups are too different, then it will be difficult to determine whether any differences in post-test scores are due to the lecture or to other factors, such as differences in the groups' baseline knowledge and skills.

The experimental time for both Group A and Group B is from 3:00 to 4:00 PM.

For group A, a pre-recorded video lecture on the internal control flow of Linux system calls was played. Group A learned based on the video content and Linux kernel source code.

For Group B, the subjects wore VR devices and entered a virtual city environment to listen to the audio lecture (the audio content was the same as group A). A teacher was responsible for explaining the corresponding relationship between various elements in the city and the components of the Linux kernel, then guiding the students into the correct building. Subjects could freely explore the virtual city in scene mode and observe various components of the city; they could also take a car in car perspective to experience process creation; in panoramic perspective they can also observe the car driving routes and understand the connections between different functions.

The second test for Group A and Group B was held from 8:00 PM to 9:00 PM. Thus, the experiment was over.

C. Analysis of Results

In the first test, the questionnaire distributed to everyone contained the following:

1) What is a system call?
2) How do Linux system calls work?
3) What are some common Linux system calls?
4) What are the advantages and disadvantages of system calls?
5) Describe the function and parameters of the fork() system call.
6) And so on.

The total score of the first survey was 100 points. The average scores and variances of Group A and Group B are shown in Fig. 13.

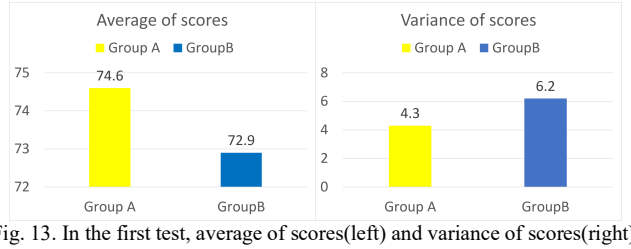

Fig. 13. In the first test, average of scores(left) and variance of scores(right).

As can be seen from Fig. 13, the average scores of Group A and Group B are close, and the variances are approximate,

indicating that the abilities of Group A and Group B are almost the same.

In the second test, the questionnaire distributed to everyone contained the following:

1) Explain the internal control flow of the fork() system call.
2) Explain the internal control flow of the exec() system call.
3) Explain the internal control flow of the munmap() system call.
4) And so on.

The total score of the second survey was also 100 points. The results of the second tests are shown in Fig. 14.

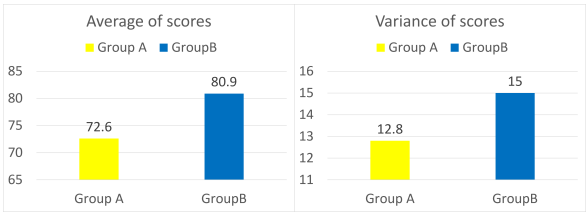

Fig. 14. In the second test, average of scores(left) and variance of scores(right).

As can be seen from Fig. 14, the average score of Group B is 8.3 higher than that of Group A. This indicates that using VR city can help students better understand the Linux kernel source code.

In addition, Group B filled out an additional questionnaire, which contained the following:

Did the everyday knowledge of the city life help in enhancing the understanding of the Linux kernel?

The survey results are shown in Fig. 15.

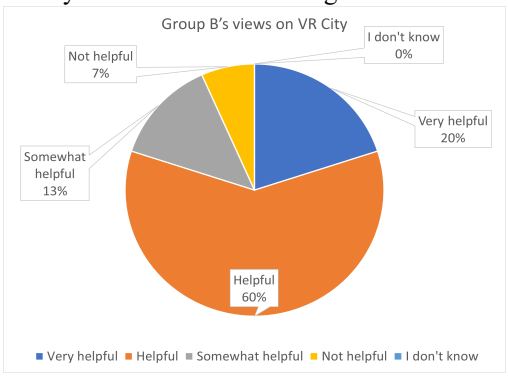

Fig. 15. In additional testing, Group B's opinion of VR City's help in understanding Linux kernel code.

V. Conclusion

Linux kernel code is massive. When explaining the internal control flow of Linux system calls using conventional teaching methods, such as reading source code, learners often get bogged down in the abstract and complex code logic, which increases the learning cost and even leads to deviations in the understanding of the control flow. In this paper, we propose VR City, which provides a virtual city environment to help learners wearing VR devices understand the internal control flow of Linux system calls. The key idea of VR City is to map the abstract and complex code logic to the city traffic that everyone is very familiar with in their daily lives, thereby reducing learners' cost of studying unfamiliar code. Our study has shown that VR City does indeed reduce the probability of learners getting lost when understanding the Linux kernel, and help learners comprehend the Linux kernel using the constructivist approach.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Shuanghui Yi conducted the research. Xiaoyan Su and Shaobo Yu analyzed the data. Bingru Wu wrote the paper. Tong Wang revised and formatted the manuscript. Hang Li coordinated the research and provided critical feedback. All authors approved the final version of the manuscript.

References

[1] Shi, J., Ji, W., Zhang, J., Gao, Z., Wang, Y., & Shi, F. (2018). Kernel Graph: Understanding the kernel in a graph. Information Visualization, 18(14), 283–296. https://doi.org/10.1177/1473871617743239

[2] Burdea, G., & Coiffet, P. (2003). Virtual reality technology (2nd ed., ch. 2). John Wiley & Sons, Inc.

[3] Marougkas, A., Troussas, C., Krouska, A., & Sgouropoulou, C. (2023). Virtual reality in education: A review of learning theories, approaches and methodologies for the last decade. Electronics, 12(13), 2832. https://doi.org/10.3390/electronics12132832

[4] Wettel, R., & Lanza, M. (2007). Visualizing software systems as cities. In 2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis (pp. 92–99). https://doi.org/10.1109/VISSOF.2007.4290706

[5] Milenković, A., Takači, Đ., & Božić, R. (2020). On the influence of software application for visualization in teaching double integrals. Interactive Learning Environments, 30(7), 1291–1306. https://doi.org/10.1080/10494820.2020.1719164

[6] Artho, C., Pande, M., & Tang, Q. (2019). Visual analytics for concurrent Java executions. In 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 1102–1105). https://doi.org/10.1109/ASE.2019.00112

[7] Davies, M. (2011). Concept mapping, mind mapping and argument mapping: What are the differences and do they matter? Higher Education, 62(3), 279–301. https://doi.org/10.1007/s10734-010-9387-6

[8] Mauerer, W. (2008). Professional Linux kernel architecture (ch. 1). Wrox.

[9] Bannert, M. (2002). Managing cognitive load—recent trends in cognitive load theory. Learning and Instruction, 12(1), 139–146. https://doi.org/10.1016/S0959-4752(01)00021-4

[10] Paas, F., Renkl, A., & Sweller, J. (2003). Cognitive load theory and instructional design: Recent developments. Educational Psychologist, 38(1), 1–4. https://doi.org/10.1207/S15326985EP3801_1