Features of Data Processing Using the Virtual File System

Svitlana Popereshnyak[1] , Serhiy Panchenko[1], Oleksii Fedorchenko[2] and Serhii Ilyin[2]

[1] *National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", 37, Prospect Beresteiskyi, Kyiv, 03056, Ukraine*

[2] *Institute of Software Systems of NAS of Ukraine, Academician Glushkov Avenue, 40, Kyiv, 03187, Ukraine*

Abstract

The work developed and considered a software system aimed at creating a virtual file system (VFS) based on FUSE, which provides a convenient and flexible interface for interacting with the file structure. The main goal of this project is to speed up file processing by customizing file system operations and creating a tool that allows users to easily manipulate the virtual file structure using a console interface. The development is intended for data processing and storage on operating systems of the Unix family. The functional tasks of the software system are focused on creating a virtual file system based on FUSE with the aim of providing users with a convenient interface for interacting with the file structure. The main goal is to provide functionality for creating and deleting objects, obtaining detailed information about file attributes, reading and writing file contents, working with symbolic links, and navigating and interacting with directories. This includes the ability to create new files and directories, delete objects, get detailed information about file attributes, read and edit their contents, and work with symbolic links. The further goal is to create a convenient and functional tool for users, which can be used for conducting experiments, training or solving specific tasks, while providing flexibility and efficiency of interaction with the FUSE-based file system through the console interface.

Keywords

Virtual file system, UNIX, FUSE, OS kernel, file, directory, soft-link, static polymorphism, hybrid systems, survivability, databases, multicriteria optimization, data center

1. Introduction

Taking into account the rapid development of modern technologies and the growth of data volumes, there is a need to improve existing virtual file systems or develop new ones that meet modern requirements for efficiency, security and flexibility. The relevance of writing a new file system lies in the ability to respond to challenges associated with increased data volume, distributed computing, privacy requirements, and high-speed information processing.

After reviewing successful analogs, it is important to note that many of them are closed source or require developers to implement their own operations, which can often limit flexibility and extensibility. In this context, a self-developed file system based on the FUSE library stands out among its peers, offering a number of significant advantages. It is not only flexible thanks to the standard FUSE interface, but it is also open source, which encourages active participation of developers and accelerates the detection and correction of errors. Additionally, confidentiality, high performance, data integrity in a multi-threaded environment, compatibility with Unix-like systems, and the ability to expand functionality by customizing operations make it an important solution for today's file system requirements.

14th International Scientific and Practical Conference from Programming UkrPROG'2024, May 14-15, 2024, Kyiv, Ukraine

* Corresponding author.

† These authors contributed equally.

spopereshnyak@gmail.com (S. Popereshnyak); panchenko.serhii@lll.kpi.ua (S. Panchenko); xmmmir@gmail.com (O. Fedorchenko); sergey.ilin.gsmart@gmail.com (S. Ilyin)

0000-0002-0531-9809 (S. Popereshnyak); 0009-0000-8897-8182 (S. Panchenko); 0009-0006-2556-7574 (O. Fedorchenko); 0009-0004-3760-1464 (S. Ilyin)

The goal of the work is to speed up file processing by customizing file system operations and creating a tool that allows users to easily manipulate the virtual file structure using a console interface.

This work will consider the concept of virtual file systems, analyze existing solutions, and justify the need to develop a new file system that meets modern requirements and takes into account current trends in the field of data processing and storage.

2. Virtual file system and fields of practical application

File systems offer a common interface for programs to access data. Although microkernels implement user-space file systems [1], most file systems are part of monolithic kernels [2]. Kernel implementations allow you to avoid large overheads for the transmission of messages of microkernels and user space daemons [3]. However, user space file systems have grown in popularity in recent years for four reasons.

1. Multiple stackable file systems add specialized features to existing file systems (such as deduplication and compression [4]).
2. In the academic environment and research institutions, this structure allowed for rapid experimentation and creation of prototypes of new approaches [5, 6].
3. Several existing kernel-level file systems have been moved to user space (eg, ZFS [7], NTFS [8]).
4. More companies are relying on user space implementations: IBM's GPFS and LTFS, Nimble Storage's CASL, Apache's HDFS, Google File System, RedHat's GlusterFS, Data Domain DDFS [9], etc.

The increasing complexity of file systems is a contributing factor to the growing popularity of file systems in user space. User space code is easier to develop, port, and maintain. Kernel errors can bring down entire systems, while user-space errors are more limited. Many libraries and programming languages are available in user space on different platforms.

A virtual file system (VFS) is a software or hardware design that provides an interface for working with a file system. It can be an abstraction layer between software and physical media that allows files and folders to be accessed regardless of their location or storage format.

VFS can provide some additional functions, such as encryption, access control, caching, compression, mapping of remote resources, etc. It allows programs to interact with the file system without taking into account the specific details of physical devices or network resources.

The main advantages of virtual file systems:

- abstraction from hardware solutions: VFS provide abstraction from specific hardware characteristics, which allows programs to work with files regardless of physical devices or their formatting; it simplifies interaction with data and allows easier transfer of programs between different systems;
- management of a variety of data sources: virtual file systems can combine data from various sources, such as local drives, network devices, cloud storage, etc., creating a convenient interface for accessing various resources;
- additional functionality: VFS can provide additional functionality, such as encryption, access control, caching, compression, and others; it allows you to expand the possibilities of working with data and provide an additional level of security;
- support of remote resources: VFS can allow access to remote resources through the network, which is important in today's world, where work with remote servers and cloud storage is becoming more and more common;

- convenience of development and testing: development and testing of programs can be simplified thanks to the use of VFS; they allow you to emulate various scenarios of interaction with the file system without binding to a specific hardware or network environment.

A virtual file system intercepts system calls related to file system operations. These calls are generated by the operating system or programs trying to interact with the file system. VFS interacts with the kernel of the operating system to process system calls and access resources. It registers in the kernel as a file operations handler and can interact with other components of the operating system. Let's consider the scheme of work on the example of Figure 1 [10].

The implementation of a virtual file system can encounter a number of problems that developers must take into account when creating an interface.

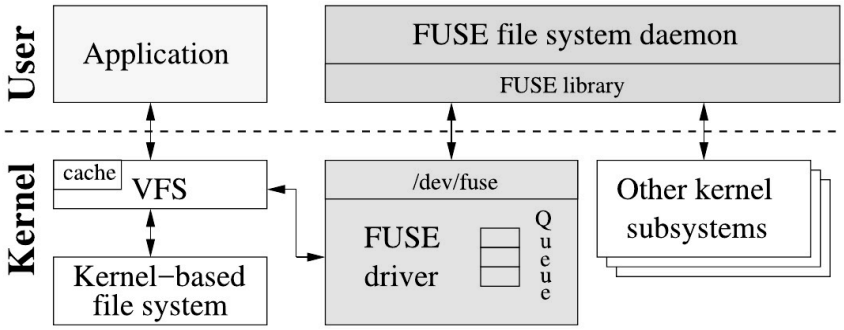

Figure 1: Scheme of the file system.

Some of the main problems include:

performance: virtual file systems can affect performance because they add an additional layer of abstraction; the efficiency of working with files and folders should be properly optimized;

security: ensuring data security and preventing possible attacks on the virtual file system is an important task; this includes access control, information encryption and other security measures;

compatibility: it is important to consider the compatibility of the virtual file system with various operating systems and programs; the development of such systems must take into account various standards and protocols;

rollback and rollback: if an error or failure occurs, it is important to have effective rollback and rollback mechanisms to prevent data loss or file system corruption;

scalability: when working with a large number of files and volume of data, it is important to ensure effective scalability of the virtual file system.

Features of processing and saving data using a virtual file system can be a key element for increasing the survivability of hybrid database management systems in the face of disruptive external influences. The use of virtual file systems can provide flexibility and reliability of data storage, which becomes especially important in scenarios where database management systems must withstand various types of load, such as cyber-attacks, natural disasters or technical failures.

Virtual file systems can effectively replicate and back up data, ensuring its availability in disaster recovery environments. For example, the ability to automatically create backup copies of data on virtual disks distributed across different physical devices or cloud services can ensure quick system recovery and minimize information loss in case of unforeseen situations.

In addition, the use of virtual file systems can allow hybrid database management systems to perform backup and data migration operations without interruption, which improves their survivability and availability.

Therefore, the use of virtual file systems can become an important element of the strategy for ensuring the survivability of hybrid database management systems in conditions of negative external influences.

Also, data processing and storage with the help of a virtual file system play an important role in the multi-criteria optimization of the load balancing of data processing centers with the variability of the intranet network topology.

Virtual file systems can help effectively manage the distribution of data between data centers, ensuring optimal availability and performance when the intranet topology changes. For example, they can automatically distribute the load based on the available bandwidth and resources of each data center.

In addition, the use of virtual file systems can help ensure that data is reliably stored in different locations, which can be important for ensuring data backup and recovery in the event of a disaster.

Therefore, the integration of virtual file systems into data management systems can contribute to the optimization of load balancing of data processing centers under conditions of intranet topology variability.

3. The brief overview of publications

Questions about virtual file systems intrigue many scientists. A file system is a set of rules that dictates how files are labeled, maintained, and accessed from a storage medium. Initially, there were numerous different file systems across various servers and machines. Various file operations are permitted only between files within the same operating systems [11].

Paper [12] introduced the Virtual Volume File System, a novel approach to on-demand processing with file system semantics, integrating these principles into a flexible and powerful data pipeline for handling some of the largest 3D volumetric datasets.

There are repositories for storing files where fast and efficient storage, search, and processing of files are crucial issues. In paper [13], the proposed approach is based on analyzing typical activities performed in research laboratories of software companies and considers the specifics of working with GitHub.

Paper [14] presents a proposed architecture for a virtual file system, which allows user rights to be granted to certain parts of a scalable file.

Emerging technologies of persistent memory, such as PCM and MRAM, offer new opportunities for preserving files in memory. Traditional file system structures may need to be re-evaluated. Paper [15] introduced a framework based on a new concept, "File Virtual Address Space". Study [16] addresses the challenges of enhancing caching efficiency for DSM-CC Data Carousel BIOP messages within the Android TV Broadcast Stack Virtual File System.

4. Success and relevance of virtual file system development

The success of virtual file systems is determined by several key criteria. First of all, it is performance, measured by speed and efficiency of work under high loads. Reliability refers to the stability and ability to restore data in the event of failures. Scalability determines the ability of the system to work efficiently when the volume of data increases. Security includes protection against unauthorized access and privacy. The work is designed for intrusion detection system that includes procedures for analyzing network honeypots and procedure for systematization received data [17]. Other important aspects include compatibility with other systems, the ability to innovate and user experience, and cost of ownership, which determines the cost of maintaining and developing a file system. All these criteria are taken into account to determine and evaluate the success of virtual file systems in the modern IT environment.

The development of your own VFS, implemented on the basis of FUSE, may be relevant for a number of reasons:

flexibility and extensibility: FUSE allows you to create virtual file systems in user space without the need to change the kernel of the operating system; it provides flexibility and extensibility in development, allowing to easily adapt VFS to the specific needs of the project;
cross-OS compatibility: virtual file systems developed using FUSE can be easily portable between different operating systems, as FUSE is supported on many platforms (Linux, macOS, FreeBSD, others); it makes development more versatile and efficient;
development of functionality: FUSE-based development allows you to create your own file systems with advanced functionality that can meet specific user or project needs; it is especially useful in cases where the existing VFS do not meet the requirements;
conducting experiments and training: the development of VFS based on FUSE can serve as an excellent opportunity for training and conducting experiments in the field of system programming; an understanding of how file systems work and how they affect the interaction with the operating system can be useful for developers.

The software system under development is aimed at creating a virtual file system (VFS) based on FUSE, which provides a convenient and flexible interface for interacting with the file structure. The main goal of this project is to create a tool that allows users to easily manipulate the virtual file structure using a console interface.

The functional tasks and purpose of the software system are focused on creating a virtual file system (VFS) based on FUSE in order to provide users with a convenient interface for interacting with the file structure. The main goal is to provide functionality for creating and deleting objects, obtaining detailed information about file attributes, reading and writing file contents, working with symbolic links, and navigating and interacting with directories. This includes the ability to create new files and directories, delete objects, get detailed information about file attributes, read and edit their contents, and work with symbolic links. A further goal is to create a convenient and functional tool for users that can be used for experiments, training or solving specific problems, while providing flexibility and efficiency of interaction with the FUSE-based file system through a console interface.

The development can be applied during the development of other types of software and in everyday interaction with files.

The target audience for the FUSE-based virtual file system includes software developers, system administrators, and other IT professionals who use and integrate file systems in their projects. Also, the system should be adapted for end users who use a virtual file system for convenient management and interaction with files.

5. Software modeling and analysis

Let's use the BPMN (Business Process Model and Notation) [9] diagram to describe the business process.

Receiving feedback from the VFS is one integrated business process, so in Figure 2 we will describe it using BPMN.

Description of the feedback from the VFS:

A request for file operations begins with a call to an external program that accesses the operating system's kernel file system.

The file system of the OS kernel processes the received request. If the file path is specified by an existing file system, the robust file system transfers control to that file system. In case the path is not found or the specified file system does not exist, the robust file system will try to handle the request itself.

If the OS kernel file system recognizes that the request refers to a virtual file system, it redirects the request to the virtual file system for further processing.

The virtual file system receives the request and processes it according to its functionality. This may include accessing real file resources, interacting with other operating system kernel components, or performing additional operations provided by the virtual file system.

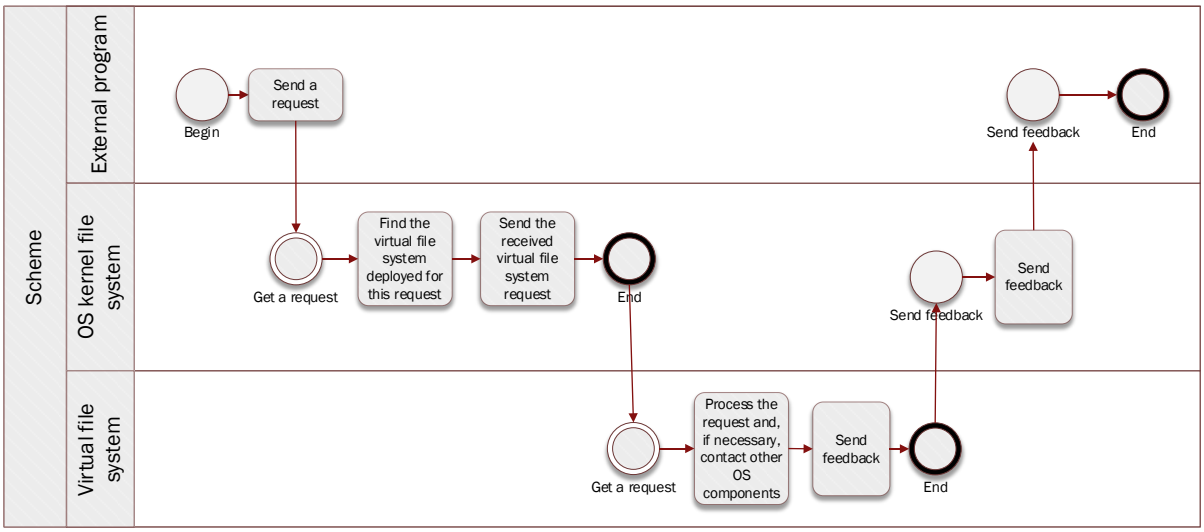

Figure 2: Scheme of the business process of receiving feedback from the VFS.

After processing the request, the virtual file system sends a response to the kernel file system, which in turn passes this response to the external application. The external program receives the result of the operation, and processing of the request is completed.

Modeling a virtual file system using BPMN allows you to clearly define its stages and the interaction of components. This facilitates analysis and optimization, and facilitates effective collaboration between development participants. The model helps identify possible risks and improve the system design, which leads to the creation of reliable and productive solutions.

5.1 Software architecture

In the development of a virtual file system, the use of design patterns becomes an important step to ensure efficiency, readability and ease of code management. One such pattern – Model-View-Controller (MVC) – allows you to separate the view, logic and data, making the architecture more modular and flexible.

The model in the virtual file system represents basic concepts such as regular files, directories, and soft links. Each of these elements corresponds to the actual structure of the file system and provides basic functionality for user interaction.

A representation decides how the information will be displayed to the user. The controller acts as an intermediary between the user and the model. It accepts commands from the user through the console and determines how those commands should be processed. After that, it interacts with the model to perform the necessary operations.

For a better understanding of the architecture, let's build a UML diagram of components. Consider the diagram of the components of the virtual file system (Figure 3.).

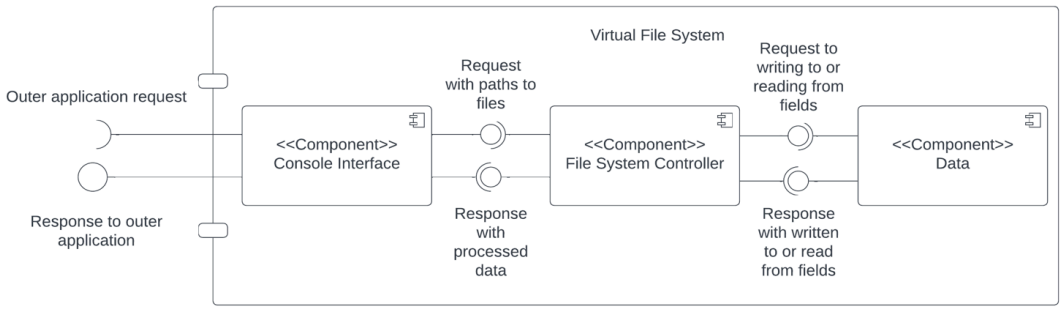

Figure 3: Diagram of the components of the VFS.

The architecture built according to the MVC pattern allows you to keep the code clean and easy to extend and modify. The distribution of responsibilities between the model, representation and controller facilitates the work on the project, ensuring optimal organization and ease of management.

5.2 Multithreading

One of the key aspects when designing a virtual file system is to ensure efficient multithreading. Since a virtual file system can be simultaneously readable and writable from different sources, it is necessary to ensure competitive and secure data access.

For this, we will use the "Adapter" pattern. The main idea of "Adapter" is to create an intermediary class (adapter) that converts the interface of one class into the interface of another class. This allows objects with different interfaces to interact with each other without direct involvement.

In this regard, the adapter allows:

ensure compatibility of class interfaces;

allow objects to work together even if their interfaces are not directly compatible.

That is, you need to build such a class, which is an adapter to ensure read-write blocking around VFS objects. This allows convenient interaction with objects, giving them the properties of mutexes.

As a result, the RwLock (Read-Write Lock) mini-library was developed. This library contains the template class TRwLock<T>, which is designed to provide competitive access to objects of type T (Figure 4).

Figure 4: Class diagram RwLock.

TrwLockGuardBase<T> is the template base class for all guards. Each child class is responsible for the specialization of the destructor, when it is called, the captured mutex is automatically released, which guarantees correct operation in a multi-threaded environment and avoidance of blocking.

5.3 Description of data structures

A detailed review of the implementation of classes and their methods will reveal the strengths and weaknesses of the data structure, analyze the optimality of the selected solutions, and solve problems that may arise when implementing a virtual file system.

Model classes, “Composite” pattern, std::variant

Consider the usual Composite pattern. This design pattern refers to structural patterns and allows clients to treat individual objects and their compositions (groups of objects) in the same way. This

pattern creates a hierarchy of classes representing both individual objects and their compositions, where both types of objects implement a common interface. Thus, the client can interact with each object (primitive or complex) through a common interface, which simplifies the processing of objects in the hierarchy.
Consider the UML diagram of file system models using the example of a virtual file system in Figure 5. On the left you can see the usual Composite, on the right - Composite combined with TRwLock:

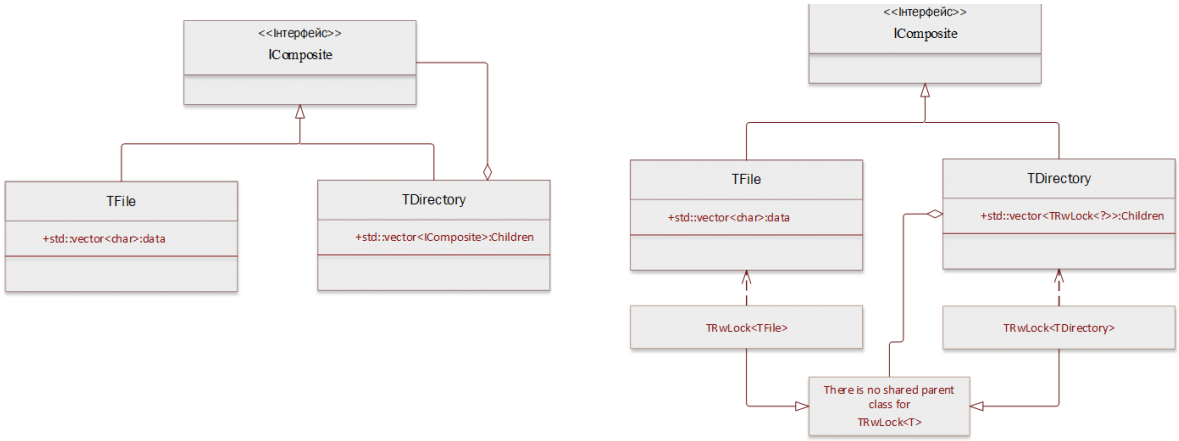

Figure 5: Composite template on the example of the VFS.

As you can see, TRwLock<TFile> and TRwLock<TDirectory> do not share a parent class, so they cannot be nested together in a container.

To solve the problem mentioned in the previous point, it is necessary to introduce a variant type that can simultaneously represent classes that do not have a common origin. For this, std::variant was introduced in C++.

6. Software quality analysis and testing

We will consider in detail the quality and stability of the developed software product, as well as conduct a testing process to ensure its reliability and efficiency.

6.1 Software quality analysis

The software quality analysis was carried out using the CCCC (C and C++ Code Counter) github application, which automatically generates a code quality report based on various metrics.

A general report of the metrics used can be seen in Table 1.

NOM (Number of modules) is the number of non-trivial modules identified by the analyzer. A total of 40 individual modules have been identified for the code.

Table 1
CCCC Code Quality Report

Metric	Tag	Overall
Number of modules	NOM	40
Lines of Code	LOC	1696
McCabe's Cyclomatic Complexity	MVG	125
Lines of Comments	COM	4

LOC (Lines of Code) — the number of non-empty lines of source code without comments, counted by the analyzer. In total, the number of lines of code is estimated at 1696 lines.

COM (Lines of Comments) — the number of lines of comments identified by the parser. In total, the number of lines of comments is estimated at 4 lines. This metric can be quite important for a large code base, but in this work the author tried to write a self-documenting code, by which a person reading can understand the meaning of what is written.

MVG (McCabe's Cyclomatic Complexity) is a measure of the complexity of the solution for the functions that make up the program. A strict definition of this measure is that it is the number of linearly independent routes through a directed acyclic graph that represents the control flow of a routine. The analyzer calculates this by recording the number of different decision outcomes contained in each function, which gives a good approximation of a formally defined version of the measure. In total, the analyzer shows 125.

6.2 Description of testing processes

The virtual file system was tested using the «googletest» testing framework. The choice of «googletest» was due to its recognized efficiency, flexibility and set of convenient tools for writing and executing tests.

One of the main advantages of «googletest» is its convenient and easy-to-use syntax for writing tests. Test cases and verifications can be easily created and organized with the help of clear mechanisms of test class structure and macros, which facilitates the process of writing and managing tests.

Another important advantage is the possibility of parallel execution of tests. This provides a faster turnaround time with a large number of test cases, which contributes to the efficiency and allocation of testing resources.

Let's show the performance evaluation, namely the speed of searching for files by their name. To do this, we will write a unit-test script that will fill the VFS with directories from A-Z, then create directories from A-Z again inside each directory, repeat this operation 4 times, that is, as a result, we will have 28 ^ 4 = 614656 directories. During testing, let's tell VFS to find all full paths of directories that have the name H (Figure 6).

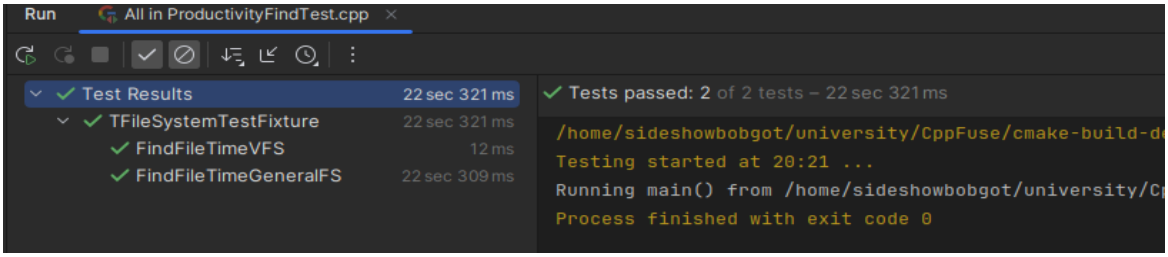

Figure 6: Search results.

We can see that the VFS found all files with the name "H" in 12 milliseconds, while the Linux file system found them in 22 seconds 309 milliseconds. Acceleration in 1859 times.

Thus, during the conducted testing, it was found that there is absolutely no validation of file ownership rights within the VFS. The Write method also malfunctioned, so it was impossible to edit files inside console text editors. It was found that sometimes VFS would go out of buffer, causing it to crash when reading links.

During manual testing, tests were conducted on the system's responsiveness to correct and incorrect data entry; saw that the system displays user-friendly error messages.

The system satisfies the condition of compatibility with other software applications, which was shown on the example of the execution of Python scripts inside the VFS.

It was also demonstrated that VFS satisfies the speed of finding files within it.

7. Conclusions

The paper examines the concept of virtual file systems, analyzes existing solutions, and substantiates the need to develop a new file system that meets modern requirements and takes into account current trends in the field of data processing and storage. During the work, a tool was created that allows users to easily manipulate the virtual file structure using a console interface, which provides opportunities to speed up file processing by customizing file system operations.

In the process of analyzing the requirements for the Virtual File System (VFS), key functional and non-functional requirements are defined. Among them are high efficiency, security and flexibility of use. The main interactions with the file system were designed, data formats were defined and the main usage scenarios were considered.

During the modeling and design phase, the architecture of the VFS was developed. It includes classes and components introduced for files and directories. Different design patterns such as Visitor are used to provide a variety of operations on different file types. Static polymorphism is implemented using templates and the CRTP pattern.

VFS testing was conducted using the «googletest» framework. It was chosen due to its high efficiency, flexibility and the possibility of parallel execution of tests. Testing covered various aspects of functionality, ensuring the stability and reliability of the software.

During the implementation phase, the steps for deploying the VFS were described in detail. This included installing the required libraries, creating a FIFO file for interposes communication, and establishing communication with the underlying file system. Documented command arguments for ease of deployment and configuration.

Future development of the VFS may include expanding functionality to support new operations, improve efficiency, and work with large amounts of data. Integration with other systems and support for new file types can become areas of further development. Continuous optimization and improvement can help the VFS improve and meet the growing requirements.

The practical significance of the developed system is as follows:

The developed file system can be applied during the development of other types of software and in everyday interaction with files.

The target audience for the FUSE-based virtual file system includes software developers, system administrators, and other IT professionals who use and integrate file systems in their projects.

The system is successfully used by system administrators who take care of servers with a significant number of files. Their tasks are related to the need to find files quickly, and this is exactly what was solved by implementing a developed file system. The use of a file system greatly improved the time of searching for files, helping to optimize the tasks of system administrators. The console interface of the system turned out to be convenient and easy to use, which facilitates the work of users.

Developers have also gained access to the open source code of the system, which allows them to customize file operations according to their needs.

In the future, it is possible to create NetFS - a distributed file system that will allow interaction with files and directories over the network.

References

[1] S. Ishiguro, J. Murakami, Y. Oyama and O. Tatebe, "Optimizing Local File Accesses for FUSE-Based Distributed Storage," 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, Salt Lake City, UT, USA (2012) 760-765. doi: 10.1109/SC.Companion.2012.104.

[2] R. Card, T. Ts'o, and S. Tweedie, Design and implementation of the second extended filesystem. In Proceedings to the First Dutch International Symposium on Linux, Seattle, WA, December 2004.
[3] M. Condict, D. Bolinger, D. Mitchell, E. McManus, Microkernel Modularity with Integrated Kernel Performance. In Proceedings of the First Symposium on Operating Systems Design and Implementation, 2014.
[4] Opendedup. URL: www.opendedup.org.
[5] J. Bent, G. Gibson, G. Grider, B. McClelland, P. Nowoczynski, J. Nunez, M. Polte, and M. Wingate, Plfs: A checkpoint filesystem for parallel applications. Technical Report LA-UR 0902117, LANL, 2009. URL: http://institute.lanl.gov/ plfs/.
[6] C. Ungureanu, B. Atkin, A. Aranya, S. Gokhale, S. Rago, G.Calkowski, C.Dubnicki, and A.Bohra. HydraFS: a High-Throughput File System for the HYDRAstor Content-Addressable Storage System. In Proceedings of the FAST Conference, 2010.
[7] ZFS for Linux. URL: www.zfs-fuse.net.
[8] NTFS-3G. USENIX Association 15th USENIX Conference on File and Storage Technologies
[9] K. Vangoor, V. Tarasov, E. Zadok, To FUSE or Not to FUSE: Performance of User-Space File Systems Bharath, Proceedings of the 15th USENIX Conference on File and Storage Technologies (2017) 59-72.
[10] FUSE - The Linux Kernel Documentation. (2023). URL: https://www.kernel.org/doc /html/ next/filesystems/fuse.html
[11] S. Parekh, A. Deshpande and N. N. Prasanth, "Time Administration of Virtual File System Operations," 2022 2nd International Conference on Technological Advancements in Computational Sciences (ICTACS) (2022) 555-558, doi: 10.1109 /ICTACS56270.2022.9987922.
[12] A. Wetzel, J. Bakal, M. Dittrich, A Virtual File System for On-Demand Processing of Multidimensional Datasets. XSEDE (2016) 35:1-35:5. doi: 10.1145/2949550.2949656.
[13] O. Chebanyuk (2023). Software Reuse Approach Based on Review and Analysis of Reuse Risks from Projects Uploaded to GitHub. In: Zlateva, T., Tuparov, G. (eds) Computer Science and Education in Computer Science. CSECS 2023. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 514. Springer, Cham. https://doi.org/10.1007/978-3-031-44668-9_11
[14] H. Sparenberg, A. Schmitt, R. Scheler, S. Foessel, K. Brandenburg, "Virtual file system for scalable media formats: Architecture proposal for managing and handling scalable media files," 2011 14th ITG Conference on Electronic Media Technology, Dortmund, Germany, (2011) 1-5.
[15] E. H.-M. Sha, X. Chen, Q. Zhuge, L. Shi and W. Jiang, "A New Design of In-Memory File System Based on File Virtual Address Framework," in IEEE Transactions on Computers, vol. 65, no. 10 (2016) 2959-2972. doi: 10.1109/TC.2016.2516019.
[16] P. Petković, T. Valeev and I. Bašičević, "Enhancing Caching Efficiency of DSM-CC Data Carousel BIOP Messages for Android TV Broadcast Stack Virtual File System," 2024 Zooming Innovation in Consumer Technologies Conference (ZINC), Novi Sad, Serbia (2024) 209-212. doi: 10.1109/ZINC61849.2024.10579367.
[17] S. Popereshnyak, O. Suprun, O. Suprun, T. Wieckowski, "Intrusion detection method based on the sensory traps system," 2018 XIV-th International Conference on Perspective Technologies and Methods in MEMS Design (MEMSTECH) (2018) 122-126. doi: 10.1109/MEMSTECH.2018.8365716.