A Tagging System Integrated into a File System

Álvaro Alvin O. Santos
Instituto Federal Catarinense
Blumenau, Santa Catarina, Brazil
alvaro.alvinn@gmail.com

Ricardo de la Rocha Ladeira
Instituto Federal Catarinense
Blumenau, Santa Catarina, Brazil
ricardo.ladeira@ifc.edu.br

Eder Augusto Penharbel
Instituto Federal Catarinense
Blumenau, Santa Catarina, Brazil
eder.penharbel@ifc.edu.br

ABSTRACT
Filesystems are essential components in operating systems. They are responsible for managing and organizing data on storage devices. This organization tends to be hierarchical, which imposes some limitations. One of them is the absence of semantic search, finding a file based on its content or expressions that define it. Another limitation is the number of semantic markers, because even if the filename and its directory are suggestive, how to assign more semantic values to a file, as if its content covers several topics? Aiming to offer a solution to these situations, this paper presents a filesystem with tagging, allowing searching for semantic markers and associating multiple markers to files. A bibliographical research allowed the finding of related works, which were analyzed to design the developed tool. The filesystem was implemented on Linux, with the disk simulation being performed by a file. A command interpreter was also implemented, allowing interaction with the filesystem and managing the markers associated with the files. The results obtained demonstrate the feasibility of implementation and serve as a starting point for filesystems to adopt this practice.

CCS Concepts

• Information systems → File systems;• Information systems → Metadata;• Information systems → Search systems.

Keywords
File system; Tagging system; Tagging.

1. INTRODUCTION
A File System (FS) is a component responsible for managing and organizing data storage on storage devices [1], such as hard drives and USB drives. It plays a fundamental role in the functioning of operating systems, as it enables the creation, reading, writing, and deletion of files, as well as providing a structure for the efficient organization and retrieval of data.

File systems typically provide a hierarchical structure represented in the form of a tree [2]. This characteristic offers advantages such as ease of implementation and logical organization of data, facilitating navigation and efficient searching of that data [2].

However, the growth in the volume of stored data makes the task of efficiently organizing and managing this information increasingly challenging [3]. In this scenario, a structure strictly based on directories is considered limited. The probability of fragmentation and the need to traverse longer paths to access files can, in certain circumstances, lead to negative performance impacts [2].

One approach that makes searching through large amounts of data easier is tagging [4], which consists of associating keywords (or tags) with objects, establishing a relationship with their content. This makes it possible to search for a file not only by its name, but also by keywords linked to it. According to [4], tagging has emerged as a recent alternative for organizing resources semantically, gaining popularity over time. A very popular example of the use of a tagging system is email tags, as can be done in Gmail (Figure 1).

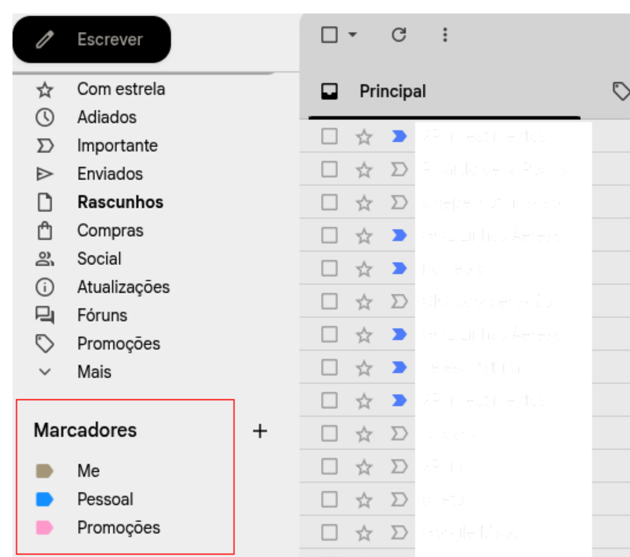

Figure 1: Gmail labels. Source: prepared by the authors.

The use of tags associated with files represents a new way of searching, allowing the file to be found even if its name is unrelated to its content. A strategy that uses multiple tags allows a file to be associated with several subjects. In a purely hierarchical system, every file is placed in a directory, and even if the directory name has some meaning, the problem remains for files on diverse subjects. Alternative solutions involve creating copies of the file, which generates waste, or creating symbolic links, which generate additional complexity, and even then, the files originally remain in only one directory.

Given the issues presented, this paper proposes an architecture for a file system with an allocation table and an integrated tagging system. The tool was developed at the user level (user space) to proving the viability of the alternative, and the allocation table was developed by the authors, not being directly related to the implementation of existing file systems, such as those of the FAT (File Allocation Table) family. The justification for this is to guarantee independence from pre-existing structures and greater control over block allocation.

Therefore, the objective of this work is to demonstrate the feasibility of implementing a file system with tagging. To achieve this, the construction of a file system based on an allocation table is presented, as well as the development of an internal tagging architecture integrated into the file system. In addition, a command interpreter was developed to demonstrate and validate the system.

This text is divided into seven sections, including this introduction. Section 2 discusses the literature review and related works, comparing them to the proposed solution. Section 3 describes the materials and methods used in the work. Section 4 discusses the design and implementation of the proposed system. Section 5 presents the results obtained, accompanied by a brief analysis. Section 6 concludes the article and presents perspectives for future work, and Section 7 contains the references used.

2. THEORETICAL BACKGROUND

In computing, data is stored as binary digits (zeros and ones), representing the smallest unit of information processable by digital devices. Permanent (non-volatile) storage is performed on devices such as hard disk drives (HDDs) [1] and solid-state drives (SSDs), which allow persistent reading and writing [5].

To manage the data stored on these devices, the operating system provides an abstraction layer called File System (FS), responsible for organizing, addressing, and controlling access to files [2].

Each file is considered a logical unit of storage that groups data under a name and a set of attributes, such as permissions, dates, and size [1]. To structure these files, operating systems use directories, which store references to other files and directories, forming a hierarchical tree structure [2]. This hierarchy allows for the logical organization of data, but restricts the association between files and multiple contexts.

According to [3], this structural limitation of hierarchical organization becomes more evident as the volume and diversity of data increase, requiring complementary organizational approaches. One such approach is tagging, which is based on associating keywords (tags) with files, allowing content to be searched using related terms.

In [4], it is highlighted that tagging introduces a semantic aspect to file management: the relationship between metadata and the meaning of the content. This brings them closer to meaning-based search systems. The use of this technique enables cross-cutting and more flexible searches, especially when implemented in conjunction with efficient data structures, such as balanced trees [6].

Thus, integrating a tagging system directly into a traditional file system represents a natural evolution in the way data is stored and retrieved, allowing the combination of the existing hierarchical structure with more modern semantic organization mechanisms.

2.1 Related Works

Several studies have proposed ways to incorporate semantic aspects into file systems, exploring the use of tags and metadata as classification and search mechanisms.

TagFS [4] is a file system that uses tags to establish semantic relationships between documents. It is based on a layer implemented over the WebDAV protocol, where tags are implicitly derived from the file paths on the server. This approach, however, limits the explicit addition and removal of tags, restricting management flexibility.

Another relevant project is the Semantic File System (SFS) [7], which introduces the concept of a semantic file system. In this model, files are described by key-value fields specific to each file type, forming metadata used in queries. The system creates virtual directories from searches, representing results as if they were real directory structures. This approach increases flexibility but heavily depends on external programs responsible for filling in the metadata fields, which reduces independence and increases the complexity of use.

hFAD (Hierarchical File Systems Are Dead) [3] is an architecture that eliminates the traditional hierarchy and organizes data in an object-oriented manner with a semantic database structure. This proposal adopts unique identifiers and metadata to represent the content, similar to Object-based Storage Devices (OSDs). The work proposes the architecture but does not present an implementation.

Unlike the approaches mentioned, the developed system integrates the explicit tagging mechanism directly into the allocation table-based file system. As a result, the relationships between files and tags become part of the file system structure itself, eliminating external metadata layers and reducing dependence on intermediary applications. This integration helps maintain more consistent associations and makes them less susceptible to failures resulting from external tools. Table 1 summarizes the differences between the related works.

Table 1: Comparison between related works.

Tool	**Method**	**Semantic approach**	**Semantic structure**	**Implemented?**
TagFS [4]	File server	*Tags*	Implicit tags	Yes
SFS [7]	Interface over FS	Key-value fields	Custom programs to fill in fields.	Yes
hFAD [3]	New FS	OSD	Database	No
This tool	Integration with SA	*Tags*	Explicit tags	Yes

3. MATERIALS AND METHODS

The operating system used for the implementation was Linux with kernel version 6.5.7. The programming language used was C, compiled with gcc version 13.2.1. The set of libraries used in building the system includes:

- **stdlib.h**: for accessing memory manipulation functions;
- **stdio.h**: for accessing input and output functions;
- **string.h**: for accessing string and memory manipulation functions;
- **stdint.h**: for accessing integer-related macros;
- **windows.h**: for system calls compatible with the Windows environment;
- **unistd.h**: for system calls compatible with the Linux environment;
- **locale.h**: for defining locale and character encoding in text output; and

- **time.h**: for accessing time functions.

In addition to the libraries mentioned, the math library (**math.h**) and the Portable Operating System Interface – POSIX threads library (**pthreads.h**) were also used, linked to the code through the -l parameter of gcc.

Regarding the research methodology, this work is characterized by experimental and bibliographic methods. The experimental method was used to verify the feasibility of integrating the file system and the tagging module. This verification occurred through the execution of commands implemented in the developed interpreter, simulating real operations of creation, reading, writing, and tag association. The bibliographic method, already explored in Section 2 (Theoretical Background) and further detailed in Section 2.1 (Related Work), investigated references that addressed the use of tagging in file systems, finding current alternatives for this functionality.

The development method adopted did not follow the classic models of Software Engineering, but blended characteristics of existing models. In this sense, the iterative and incremental approaches [8] stand out, as each new functionality was incorporated into the main product developed, so that the proposed solution was fully integrated at the end. The main functionalities were developed first and refined in incremental cycles, which reduced inconsistencies throughout the iterations.

The management of the most recent versions of the system was carried out using a repository[1] on GitLab, where the most up-to-date source code, the makefile used, and usage instructions can be found.

4. DESIGN AND IMPLEMENTATION

The project was structured into three main components: the base file system, the tagging system, and the integration module between them, and a command interpretation component for interacting with the FS and managing the tags.

The base FS was developed based on the architecture proposed in [2]. It uses an allocation table that represents the blocks of the storage device, similar to the FAT approach, but implemented independently, without fixed reserved sectors and with mapping controlled directly in memory, which reduces the need for disk movements and facilitates debugging of the structure during development. Each index in the table corresponds to a logical block and stores the pointer to the next block, forming a linked list. This structure simplifies space management and facilitates sequential reading of files.

Figure 2 illustrates the logical structure of the allocation table used, showing how each block of the device is linked to the next, forming a linked list that defines the sequence of data storage.

Allocation table	
Index	Value
0	3
1	0
2	5
3	4
4	-1
5	-1
...	...
N	0

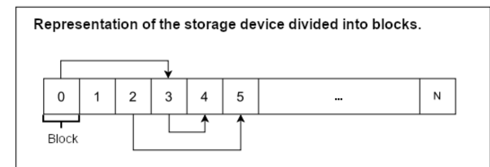

Figure 2: File allocation table. Source: prepared by the authors.

The tagging system was implemented based on an AVL tree, ensuring efficiency in search, insertion, and deletion operations. Each node of the tree contains a linked list of file identifiers, allowing multiple associations per tag. The use of this balanced structure ensures that access time remains in the order of *O(log n)*, where *n* represents the total number of nodes in the AVL tree, i.e., the total number of tags stored in the system. Access time will remain logarithmic even with an increase in the number of tags [6]. The choice of the AVL tree is due to its simplicity of implementation and the balance between search speed and restructuring cost, suitable for the context of system development at the user level.

Figure 3 presents the data structure of the tagging system, where each node of the tree represents a tag and contains a list of associated files.

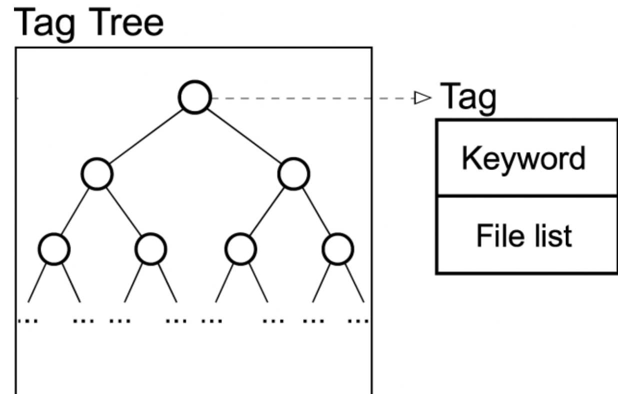

Figure 3: Structure of the tagging system. Source: prepared by the authors.

Each tag maintains a linked list of file identifiers representing all the files associated with it. This structure allows for multiple associations in a simple and dynamic way, ensuring that a search for a tag returns all corresponding files. Figure 4 illustrates the structure of this association list.

[1]https://gitlab.com/alvaro.alvinn/so-sistema-de-arquivos

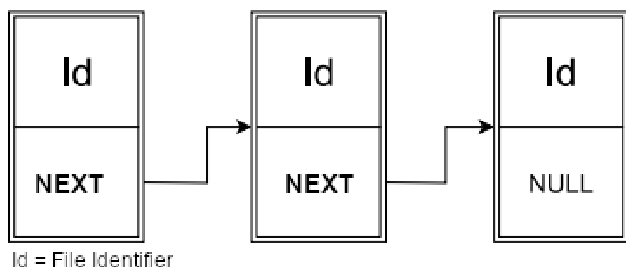

Figure 4: Linked list of file identifiers associated with a tag. Source: prepared by the authors.

The integration between the two components of the system was designed to preserve the integrity of the associations between files and tags. Each file has a header containing the following metadata:

- name
- size
- creation date
- pointer to the first data block
- parent directory identifier
- list of associated tags

This structure is read whenever the file is opened, allowing quick access to association information without the need to traverse the entire tag tree. The tags and their associations are also stored in the tag tree, ensuring redundancy and enabling the recovery of links in cases of failures or inconsistencies. Both the tag tree and the association lists can grow dynamically, being written in chained blocks as the volume of data increases.

To allow interaction with the system, a command interpreter was developed that simulates a terminal. It offers instructions similar to those of Unix systems:

- **cat**: prints the contents of a file.
- **echo**: writes to a text file.
- **ls**: list the contents of the current directory.
- **mkdir**: creates an empty directory in the current directory.
- **rm**: deletes a file or directory.
- **touch**: creates a blank file in the current directory.
- **cd**: changes the current directory.
- **addtag**: adds a new tag (first parameter) to a file (second parameter).
- **rmtag**: removes a tag (first parameter) from a file (second parameter).
- **tagfind**: lists the complete path of all files associated with the tag provided as a parameter.
- **namefind**: searches for an expression specified by a parameter recursively from the current directory, displaying all found files that contain the searched term in their name.
- **ltags**: lists the tags that are associated with a file whose name is passed as a parameter.
- **ctags**: removes all tags associated with a file whose name is passed as a parameter.
- **find**: executes the tagfind and namefind commands simultaneously listing the found files; however, in this way, the files found based on the tags are identified with the expression "(TAG)" written next to them.

This component is not part of the core file system, but acts as a testing and validation interface, allowing the creation, listing, and searching of files and their tags interactively. This made it possible to verify the correct functioning of the read, write, and tag association operations.

The result is a functional system capable of performing read, write, and search operations by name or tag in an integrated manner, demonstrating the technical feasibility of the integration. The implemented system was subjected to a series of functional tests cited in the following section, in order to evaluate its behavior.

5. RESULTS ANALYSIS

The tests conducted involved the creation and manipulation of multiple files with different numbers of tags, simulating real-world usage scenarios, and demonstrated the proper functioning of the FS integrated into the tagging module. The developed command interpreter simulated an operating system terminal, allowing the execution of operations such as creating and removing files and directories as well as associating, listing, and searching for tags.

Figures 5 and 6 show how the tags are registered in the file and directory blocks, along with the metadata information, indicating the keywords associated with the content.

Although the focus of this work is not performance optimization, it was observed during the experiments that the execution time of the operations remained within the expected range for a system based on an allocation table. The integration of tags did not generate noticeable overhead in simple operations, although the serialization of the tag tree to disk, performed with each modification, impacted the response time in intensive usage scenarios. Thus, operations involving the serialization of the tree indicate opportunities for future optimization.

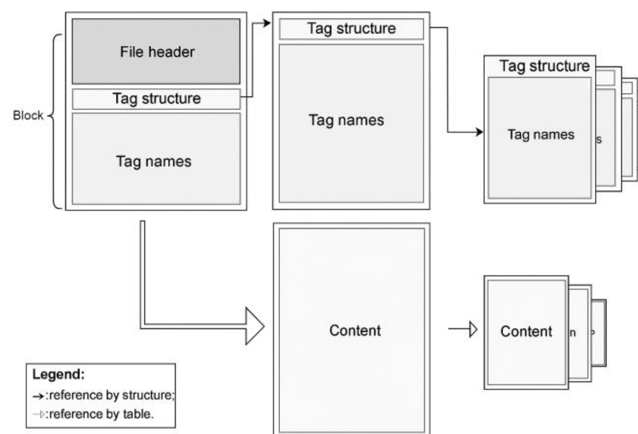

Figure 5: Tag structure in files. Source: prepared by the authors.

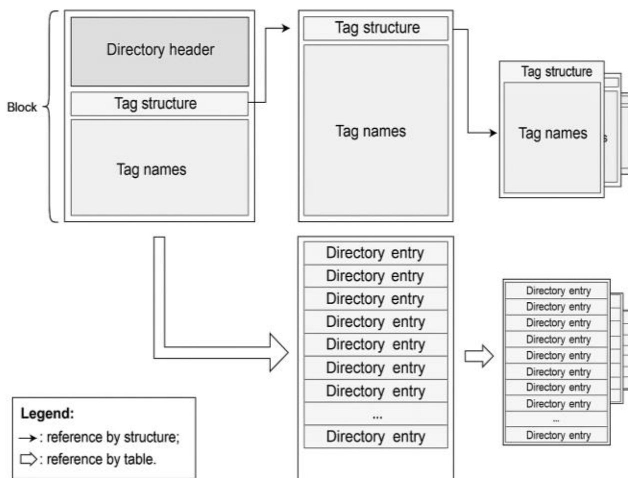

Figure 6: Tag structure in directories. Source: prepared by the authors.

Nevertheless, the results confirm that associating tags directly with the file system is feasible, both in terms of structure and functionality. The redundancy of information between files and the tag tree contributed to the integrity of the system, allowing recovery in simulated cases of partial data corruption.

The system's behavior was also verified in tag removal and reassignment operations. In all cases, the associations were correctly updated in both the tag tree and the file headers, preserving structural consistency even after multiple modifications.

Frame 1 presents an example of executing commands in the developed interpreter, illustrating the creation of files, the association and removal of tags, and the subsequent verification of the updated associations. The string "/$" represents the terminal prompt from the / (root) directory.

Frame 1: Example of executing commands in the interpreter, with the addition and removal of tags. Source: prepared by the authors.

```
/$ touch example.txt
/$ addtag article example.txt
/$ addtag system example.txt
/$ addtag work example.txt
/$ ltags example.txt
article
system
work
/$ rmtag work example.txt
/$ ltags example.txt
article
system
/$ ctags example.txt
/$ ltags example.txt
```

These results validate the hypothesis that file systems can natively incorporate semantic mechanisms, paving the way for more efficient and integrated future solutions.

6. CONCLUSION

This paper demonstrated the feasibility of implementing a file system containing a tagging system. The use of tags provided an additional means of searching for files, in addition to the traditional directory-based method, allowing for semantic associations between files and content. Experimental results demonstrate that it is possible to integrate the tagging functionality directly into the file system, eliminating the need for additional configurations or external tools. The practical verification of this integration was carried out through the developed command interpreter, which reproduces typical file system terminal operations and demonstrated the correct functioning of the system.

Future work includes optimizing memory usage and disk access, especially with regard to the tag tree. The tree is fully written to disk with each modification, which is computationally expensive. Among the planned improvements is the implementation of a journaling system, a component present in modern systems. Thus, operations can be recorded, tracked, and performed more conveniently.

There are several ways to extend this work. One is to modernize the addressing system using inodes, allowing decentralized allocation tables and on-demand memory management. Another is to implement the system at the kernel level, turning it into a real file system. It is also possible to add recovery mechanisms for failures, since information is stored both in the tag tree and in tagged files. enabling one structure to repair the other and maintain consistency. Finally, an automatic tag generation mechanism based on file content could be developed.

7. REFERENCES

[1] Silberschatz, Abraham; Galvin, Peter B.; Gagne, Greg. 2018. Operating System Concepts. Wiley, Hoboken, NJ, USA.

[2] Tanenbaum, A. S.; Bos, H. 2015. Modern Operating Systems (4th ed.). Pearson Education, Harlow, UK.

[3] Seltzer, Margo I., and Murphy, Nicholas. 2009. Hierarchical File Systems Are Dead. In HotOS 2009.

[4] Bloehdorn, Stephan, et al. 2006. TagFS – tag semantics for hierarchical file systems. In Proceedings of the 6th International Conference on Knowledge Management (I-KNOW '06), Graz, Austria. 6-8.

[5] Stallings, William. 2018. Operating Systems: Internals and Design Principles (9th ed.). Pearson, Boston, MA, USA.

[6] Standish, Thomas A. 1980. Data Structure Techniques. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[7] Gifford, D. K., et al. 1991. Semantic file systems. ACM SIGOPS Operating Systems Review, 25, 5 (Sept. 1991), 16-25.

[8] Pressman, Roger S., and Maxim, Bruce R. 2021. Engenharia de Software-9. McGraw Hill Brasil, São Paulo, Brazil.