

MA1-IRELE

**OS and security**

---

# CmOS

---

*Author :*

Emmeran Colot

*Professor :*

Prof. B. Da Silva

**Academic year :**

2024 - 2025

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Background</b>	<b>2</b>
2.1	disk . . . . .	2
2.2	Paging . . . . .	2
2.3	File system . . . . .	2
<b>3</b>	<b>Description of the storage systems</b>	<b>3</b>
3.1	generalities . . . . .	3
3.1.1	Variable page size . . . . .	3
3.1.2	Folders . . . . .	3
3.1.3	Addressing bytes . . . . .	3
3.1.4	File allocation table . . . . .	3
3.1.5	File identifier . . . . .	3
<b>4</b>	<b>Experimental results</b>	<b>4</b>
<b>5</b>	<b>Conclusion</b>	<b>5</b>

In this report, a comparison of two ways of storing files is made. There is a main focus on how those storage systems work, followed by a comparison of their performance.

The project was initially named *CmOS* for *C modeled OS* as it was supposed to implement more than only a storage system. The whole code is available on github<sup>1</sup> and one can see that the basic building blocks for a complete OS are present. It includes registers definition, a custom ISA, a compiler and an interpreter for going from assembly code to binary code and then execute it and so on, but they will not be discussed as the project objective has changed to the study of storage systems. It is a smaller topic but there is still a lot to say about it.

Finally, there are a few remarks:

- This whole project has been built by hand with no external source. This means that the two systems are not based on any existing system but only on the knowledge acquired during the course and on some personal ideas.
- In the following, the terms *file system* and *storage system* will be used interchangeably
- Except when specifically mentioned, the term *disk* will refer to the simulated disk.

---

<sup>1</sup><https://github.com/e-colot/CmOS>

As everything is built from scratch, there is not a lot of prerequisite knowledge needed to understand this report. However, a few concepts are important to understand how the two systems work.

## 2.1 disk

The disk is where files are stored on a computer and it can be seen as a big vector of bytes. To read or write some data, an address is given and the driver will read or write the data at that address. A real disk will often have different sectors with a longer access time when changing sectors.

In the simulation, the disk is a file on the computer of which the size is fixed by a parameter. Because of this, there is no access time when changing sectors. However, because of paging (which will be explained later) and depending on the computer on which the simulation is run, the access time could indeed be longer when reading or writing data at addresses that are far away from each other but this is, once again, outside of the scope of this analysis.

## 2.2 Paging

When storing data in memory, it would be inefficient to have data blocks of varying sizes. This is why the memory is divided into blocks of fixed size. Each of those blocks is called a *page* and the size of a page is called the **page size**. This size will be an important parameter in the following.

## 2.3 File system

The file system is in charge of managing the data on the disk. It is responsible for writing files, accessing written files and deleting them. It might have additional features but those are sufficient for a working file system and the one built here only implement those basic operations.

## 3.1 generalities

### 3.1.1 Variable page size

As described in section 2.2, both file systems will use paging to store data. The page size is variable and can be set when creating the file system. Building a variable page size system is not a difficult task unless it aims to be efficient. Both of the systems built here have been designed to be efficient so this single feature has doubled the development time.

### 3.1.2 Folders

For the sake of simplicity, the two systems will not implement folders. This is a choice that is often made for small size operating systems such as for RTOS<sup>1</sup>.

### 3.1.3 Addressing bytes

The disk is split into pages of a fixed size but when trying to write or read a page, there must be a way of pointing to it. Because the disk size and the page size are parameters, one can not assume a single byte address will be enough to address all the pages.

A disk of 256 kB with a page size of 256 bytes will have 1024 pages and a single byte address can only go up to 255. This means that a single byte address will not be enough to address all the pages. This is why the number of bytes used to address a page, referred to as **addressing bytes**, is variable and is computed as follows:

### 3.1.4 File allocation table

### 3.1.5 File identifier

---

<sup>1</sup>Real Time Operating System



