

CSCE 611: Operating Systems

Fall 2022

Machine Problem 7: Vanilla File System

Name: Rajesh Satpathy

UIN: 931006158

Goal:

Implement a simple file system with the following functionalities and constraints:

- Files support sequential access only
- File Namespace is very simple and files are identified by unsigned integers; no multilevel directories are supported

Problem Statement:

1. Implement the simple file system for the three classes FileSystem, File, and Inode where each file has a maximum size of 512 Bytes.
2. Use the provided code in the file.H/C and file_system.H/C, which contain the definition and a dummy implementation of classes File, Inode, and FileSystem, respectively.
3. Check that the test function exercise file system() generates the correct results. The goal is not to make it throw assertion errors.
4. Pick one or more options and improve your le-system design and implementation.

Scope

- Implement the file system for points 1 to 3 in the problem statement.
- Design of an extension to the basic file system to allow for files that are up to 64kB long. (Option 1)

Implementation:

The implementation is split into file management and file system management.

File Management:

- A reference is made to the inode with an inode index and the file system with inode with a file system pointer is made.
- A current position representing the file position variable is maintained.
- Implementations are added for the File Constructor, Destructor, Read, Write, Reset, and EOF functionalities.

File System Management:

- An inode counter, a variable to track free blocks, and a bitmap implementation of free blocks are made to manage the file system.
- Implementations are added to the FileSystem's Constructor, Destructor, Mount, Format, LookupFile, CreateFile, and DeleteFile.

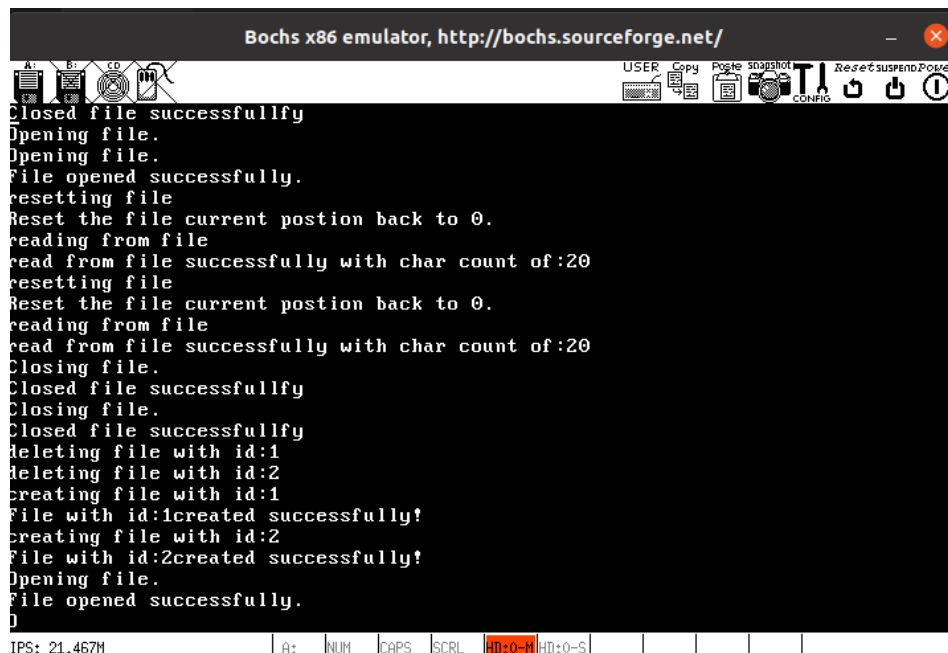
Option 1: Implementation of File System for files that are up to 64kB long.

For this scenario, we can have multiple blocks assigned to one file. To do this, we can maintain an inode pointer that keeps track of the sequence of blocks associated with the file.

For the implementation, we have to do the following updates:

- Create File: Assign a block list to the file, instead of only one block in the File System method.
- Delete File: Loop through and mark all blocks free when deleting files in the file system method.
- Read File: Sequential reading has to be updated to accommodate block list reading. The number of blocks has to be maintained and looped to read a whole file across blocks and use the EOF flag right.
- Write File: Sequential writing has to be updated similarly to the reading of the file. If there is an overflow we can introduce a link to the next block at end of the block to assign a new block list where writing can continue.

A screenshot of the output is attached below:



The screenshot shows a Bochs x86 emulator window with the title bar "Bochs x86 emulator, http://bochs.sourceforge.net/". The window contains a terminal-like interface with the following text output:

```
Closed file successfully
Opening file.
Opening file.
File opened successfully.
resetting file
Reset the file current position back to 0.
reading from file
read from file successfully with char count of:20
resetting file
Reset the file current position back to 0.
reading from file
read from file successfully with char count of:20
Closing file.
Closed file successfully
Closing file.
Closed file successfully
Deleting file with id:1
Deleting file with id:2
creating file with id:1
File with id:1 created successfully!
creating file with id:2
File with id:2 created successfully!
Opening file.
File opened successfully.
)
```

At the bottom of the window, there is a status bar with the following information: IPS: 21,467M, A:, NUM, CAPS, SCRL, HD:0-M, HD:0-S.