CSCI 430 File System Simulation: Design Document

Lucas Furlong

Davenport University

CSCI 430 File System Simulation: Design Document

**Overview**

In this assignment a basic file system prototype will be implemented. The allocation method that will be used for this file system is first fit. This design document will cover the allocation method, the pseudocode of the commands that will be implemented in the file system, and the testing that should take place for the file system to properly operate.

**Allocation Method**

The allocation method that will be implemented in this file system will be a block style method that utilizes the first fit structure. Block style allocation works by creating different sections in the allocated file system storing the directory, allocation lock units, and the storage area. The storage area will contain blocks with a specified size that can hold the data for the specific files. The allocation lock area will be an array of bytes that contains values of 0 and 1 letting the program know if a specific block is occupied with a files data. The first fit method searches the allocation lock area of the file system and finds the first available block that can store the file and puts the data in the storage area in that block.

**Pseudocode of Commands**

**Volume Commands**

This section will cover the commands that deal with allocating and mounting space for the initialization of the file system.

***ALLOCATE***

Open new filestream with user created volume name

For the length of the fixed directory size, write a null byte in the stream

Subtract number of bytes from user size

Take user size and divide by 512 (this is the number of available blocks)

Write a 0 for the number of available blocks

Subtract number of bytes from user size

If user size does not equal zero

For each block write null bytes for the size, end the block with an ETX byte, and subtract that amount of data from the user data

Close filestream ***DEALLOCATE***

If volume exists delete it ***TRUNCATE***

If volume exists

Open a filestream for the volume

Reinitialize the volume (call the allocate function) ***DUMP***

If volume exists

Open filestream for the volume

For every byte in the volume, write the byte to the screen

***CATALOG***

If volume exists

Open filestream for the volume

Return every file name in the directory area of the volume ***MOUNT***

If volume exists

Create new file system object that starts a filestream ***UNMOUNT***

If file system exists

Close filestream and dispose all resources ***INFO***

If file system exists

Return volume size, number of bytes with null value, and number of files in the directory

**File Commands**

This section will cover commands that directly affect the files that are stored within the file system after it is created. ***CREATE***

Make new directory entry and populate it with user given filename ***WRITE***

If file exists

Open filestream with specified buffer with filename specified

Find first open block on file system

If block has enough data

For every byte to the length of data entered write a byte to the file

***READ***

If file exists

Open filestream with specified buffer with filename specified

Find file block using block pointer

For every byte to the length of data entered read the byte on the file ***DELETE***

If file exists

If file is not read only

Delete file

Set block pointer to 0

Else return error message

***TRUNCATE***

If file exists

Call delete function for user specified filename

Call create function for user specified filename

***INFO***

If file exists

Return all properties available for the directory entry (these include filename, size in bytes, if file is read-only, date and time created, and date and time last modified)

***SET***

If file exists

Set read-only flag for user specified filename to true or false based on user specified entry

**Testing**

This section will cover the testing that was accomplished after the creation of the program. Each section will cover the testing of each command. Commands not implemented due to time constraints were the info command for files and the set command for files.

**ALLOCATE**

|  |  |  |
| --- | --- | --- |
| Scenario | Description | Pass/Fail |
| 1 | Test for correct args | PASS |
| 2 | Test for volume boundaries | PASS |
| 3 | Test for volume creation | PASS |

**Scenario One: Test for correct args**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Type wrong number of args for command |  |
| 3 | The program should return an error message that says, “Invalid number of arguments for the ALLOCATE command.” |  |
| EXPECTED OUTPUT | | Error message from step #3 |
| ACTUAL OUTPUT | | Error message from step #3 |
| RESULTS | | PASS |

**Scenario Two: Test for volume boundaries**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Type under 1000 bytes or over 5076 bytes |  |
| 3 | The program should return an error message that says, “Volume cannot be more than 5076 bytes!” or “Volume must be bigger than 1000 bytes!” |  |
| EXPECTED OUTPUT | | Error message from step #3 |
| ACTUAL OUTPUT | | Error message from step #3 |
| RESULTS | | PASS |

**Scenario Three: Test for volume creation**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Open file system and get to folder where volume was created |  |
| 2 | See if volume is there with correct number of bytes |  |
| EXPECTED OUTPUT | | Result from step #2 |
| ACTUAL OUTPUT | | Result from step #2 |
| RESULTS | | PASS |

**DEALLOCATE**

|  |  |  |
| --- | --- | --- |
| Scenario | Description | Pass/Fail |
| 1 | Test for correct args | PASS |
| 2 | Test for volume deletion | PASS |
| 3 | Test for volume existence | PASS |

**Scenario One: Test for correct args**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Type wrong number of args for command |  |
| 3 | The program should return an error message that says, “Invalid number of arguments for the DEALLOCATE command.” |  |
| EXPECTED OUTPUT | | Error message from step #3 |
| ACTUAL OUTPUT | | Error message from step #3 |
| RESULTS | | PASS |

**Scenario Two: Test for volume deletion**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Open file path where volume was |  |
| 2 | Check for volume deletion |  |
| EXPECTED OUTPUT | | Result from step #2 |
| ACTUAL OUTPUT | | Result from step #2 |
| RESULTS | | PASS |

**Scenario Three: Test for volume existence**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Type wrong volume name |  |
| 3 | The program should return an error message that says, “File does not exist!” |  |
| EXPECTED OUTPUT | | Error message from step #3 |
| ACTUAL OUTPUT | | Error message from step #3 |
| RESULTS | | PASS |

**TRUNCATE (VOLUME)**

|  |  |  |
| --- | --- | --- |
| Scenario | Description | Pass/Fail |
| 1 | Test for correct args | PASS |
| 2 | Test for volume empty | PASS |
| 3 | Test for volume existence | PASS |

**Scenario One: Test for correct args**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Type wrong number of args for command |  |
| 3 | The program should return an error message that says, “Invalid number of arguments for the TRUNCATE command.” |  |
| EXPECTED OUTPUT | | Error message from step #3 |
| ACTUAL OUTPUT | | Error message from step #3 |
| RESULTS | | PASS |

**Scenario Two: Test for volume empty**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Truncate volume |  |
| 3 | The program should return an empty volume from the same name |  |
| EXPECTED OUTPUT | | Result from step #3 |
| ACTUAL OUTPUT | | Result from step #3 |
| RESULTS | | PASS |

**Scenario Three: Test for volume existence**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Type wrong volume name |  |
| 3 | The program should return an error message that says, “File does not exist!” |  |
| EXPECTED OUTPUT | | Error message from step #3 |
| ACTUAL OUTPUT | | Error message from step #3 |
| RESULTS | | PASS |

**DUMP**

|  |  |  |
| --- | --- | --- |
| Scenario | Description | Pass/Fail |
| 1 | Test for correct args | PASS |
| 2 | Test for volume dump | PASS |

**Scenario One: Test for correct args**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Type wrong number of args for command |  |
| 3 | The program should return an error message that says, “Invalid number of arguments for the DUMP command.” |  |
| EXPECTED OUTPUT | | Error message from step #3 |
| ACTUAL OUTPUT | | Error message from step #3 |
| RESULTS | | PASS |

**Scenario Two: Test for volume dump**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Dump volume |  |
| 3 | The program should return the volume with all data contained inside |  |
| EXPECTED OUTPUT | | Result from step #3 |
| ACTUAL OUTPUT | | Result from step #3 |
| RESULTS | | PASS |

**CATALOG**

|  |  |  |
| --- | --- | --- |
| Scenario | Description | Pass/Fail |
| 1 | Test for correct output | PASS |

**Scenario One: Test for correct output**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Mount volume |  |
| 3 | Run catalog command and it should post all file names in the volume |  |
| EXPECTED OUTPUT | | Result from step #3 |
| ACTUAL OUTPUT | | Result from step #3 |
| RESULTS | | PASS |

**MOUNT**

|  |  |  |
| --- | --- | --- |
| Scenario | Description | Pass/Fail |
| 1 | Test for volume existence | PASS |
| 2 | Test for correct args |  |

**Scenario One: Test for volume existence**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Type wrong volume name |  |
| 3 | The program should return an error message that says, “File does not exist!” |  |
| EXPECTED OUTPUT | | Error message from step #3 |
| ACTUAL OUTPUT | | Error message from step #3 |
| RESULTS | | PASS |

**Scenario Two: Test for correct args**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Type wrong number of args for command |  |
| 3 | The program should return an error message that says, “Invalid number of arguments for the MOUNT command.” |  |
| EXPECTED OUTPUT | | Error message from step #3 |
| ACTUAL OUTPUT | | Error message from step #3 |
| RESULTS | | PASS |

**UNMOUNT/EXIT**

|  |  |  |
| --- | --- | --- |
| Scenario | Description | Pass/Fail |
| 1 | Test for work | PASS |

**Scenario One: Test for work**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Mount volume |  |
| 3 | Type exit and it should exit the volume |  |
| EXPECTED OUTPUT | | Result from step #3 |
| ACTUAL OUTPUT | | Result from step #3 |
| RESULTS | | PASS |

**INFO**

|  |  |  |
| --- | --- | --- |
| Scenario | Description | Pass/Fail |
| 1 | Test for volume existence | PASS |
| 2 | Test for correct args | PASS |
| 3 | Test for correct output | PASS |

**Scenario One: Test for volume existence**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Type info command without the right file name |  |
| 3 | The program should return an error message that says, “File does not exist!” |  |
| EXPECTED OUTPUT | | Error message from step #3 |
| ACTUAL OUTPUT | | Error message from step #3 |
| RESULTS | | PASS |

**Scenario Two: Test for correct args**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Type wrong number of args for command |  |
| 3 | The program should return an error message that says, “Invalid number of arguments for the INFO command.” |  |
| EXPECTED OUTPUT | | Error message from step #3 |
| ACTUAL OUTPUT | | Error message from step #3 |
| RESULTS | | PASS |

**Scenario Three: Test for correct output**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Type info command |  |
| 3 | The program should return the file size, number of files, and number of available bytes |  |
| EXPECTED OUTPUT | | Error message from step #3 |
| ACTUAL OUTPUT | | Error message from step #3 |
| RESULTS | | PASS |

**CREATE**

|  |  |  |
| --- | --- | --- |
| Scenario | Description | Pass/Fail |
| 1 | Test for file existence (can’t create another file by the same name | PASS |
| 2 | Test for correct args | PASS |
| 3 | Test for correct output | PASS |
| 4 | Test for filename length | PASS |

**Scenario One: Test for file existence**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Type the same file name as one that already exists |  |
| 3 | The program should return an error message that says, “File already exists!” |  |
| EXPECTED OUTPUT | | Error message from step #3 |
| ACTUAL OUTPUT | | Error message from step #3 |
| RESULTS | | PASS |

**Scenario Two: Test for correct args**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Type wrong number of args for command |  |
| 3 | The program should return an error message that says, “Invalid number of arguments!” |  |
| EXPECTED OUTPUT | | Error message from step #3 |
| ACTUAL OUTPUT | | Error message from step #3 |
| RESULTS | | PASS |

**Scenario Three: Test for correct output**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Type catalog |  |
| 3 | The program should return the catalog for the volume with the file created |  |
| EXPECTED OUTPUT | | Result from step #3 |
| ACTUAL OUTPUT | | Result from step #3 |
| RESULTS | | PASS |

**Scenario Four: Test for filename length**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Type a file length over 40 bytes |  |
| 3 | The program should return an error message that says, “File name length cannot be over 40 characters!” |  |
| EXPECTED OUTPUT | | Error message from step #3 |
| ACTUAL OUTPUT | | Error message from step #3 |
| RESULTS | | PASS |

**WRITE**

|  |  |  |
| --- | --- | --- |
| Scenario | Description | Pass/Fail |
| 1 | Test for correct args | PASS |
| 2 | Test for file correct start boundaries | PASS |
| 3 | Test for end of sector | PASS |
| 4 | Test for write | PASS |
| 5 | Test for file existence | PASS |

**Scenario One: Test for correct args**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Type wrong number of args for command |  |
| 3 | The program should return an error message that says, “Invalid number of arguments!” |  |
| EXPECTED OUTPUT | | Error message from step #3 |
| ACTUAL OUTPUT | | Error message from step #3 |
| RESULTS | | PASS |

**Scenario Two: Test for file correct start boundaries**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Type start number over 512 bytes or under 0 bytes |  |
| 3 | The program should return an error message that says, “Start position cannot be bigger than file sector size!” or “Start position cannot be under 0! |  |
| EXPECTED OUTPUT | | Error message from step #3 |
| ACTUAL OUTPUT | | Error message from step #3 |
| RESULTS | | PASS |

**Scenario Three: Test for end of sector**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Write data over file sector size |  |
| 3 | The program should return an error message that says, “Wrote, but reach end of sector. Data might have not been written all the way” |  |
| EXPECTED OUTPUT | | Error message from step #3 |
| ACTUAL OUTPUT | | Error message from step #3 |
| RESULTS | | PASS |

**Scenario Four: Test for write**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | write data |  |
| 3 | Dump volume and there should be data in the correct file sector |  |
| EXPECTED OUTPUT | | Result from step #3 |
| ACTUAL OUTPUT | | Result from step #3 |
| RESULTS | | PASS |

**Scenario Five: Test for file existence**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Write wrong filename |  |
| 3 | Program should return “No file by such name to write to!” |  |
| EXPECTED OUTPUT | | Result from step #3 |
| ACTUAL OUTPUT | | Result from step #3 |
| RESULTS | | PASS |

**READ**

|  |  |  |
| --- | --- | --- |
| Scenario | Description | Pass/Fail |
| 1 | Test for correct args | PASS |
| 2 | Test for file correct start and end boundaries | PASS |
| 3 | Test for end of sector | PASS |
| 4 | Test for read | PASS |
| 5 | Test for file existence | PASS |
| 6 | Test for no read data | PASS |

**Scenario One: Test for correct args**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Type wrong number of args for command |  |
| 3 | The program should return an error message that says, “Invalid number of arguments!” |  |
| EXPECTED OUTPUT | | Error message from step #3 |
| ACTUAL OUTPUT | | Error message from step #3 |
| RESULTS | | PASS |

**Scenario Two: Test for file correct start and end boundaries**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Type start/end number over 512 bytes or under 0 bytes |  |
| 3 | The program should return an error message that says, “Start/end position cannot be bigger than file sector size!” or “Start/end position cannot be under 0! |  |
| EXPECTED OUTPUT | | Error message from step #3 |
| ACTUAL OUTPUT | | Error message from step #3 |
| RESULTS | | PASS |

**Scenario Three: Test for end of sector**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Write data over file sector size |  |
| 3 | The program should return an error message that says, “end of sector” |  |
| EXPECTED OUTPUT | | Error message from step #3 |
| ACTUAL OUTPUT | | Error message from step #3 |
| RESULTS | | PASS |

**Scenario Four: Test for read**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Read written data |  |
| 3 | The program should return written data |  |
| EXPECTED OUTPUT | | Result from step #3 |
| ACTUAL OUTPUT | | Result from step #3 |
| RESULTS | | PASS |

**Scenario Five: Test for file existence**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Write wrong filename |  |
| 3 | Program should return “No file by such name to read from!” |  |
| EXPECTED OUTPUT | | Result from step #3 |
| ACTUAL OUTPUT | | Result from step #3 |
| RESULTS | | PASS |

**Scenario Six: Test for no read data**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Write start and end that contains no data to read |  |
| 3 | Program should return “No data found to read” |  |
| EXPECTED OUTPUT | | Result from step #3 |
| ACTUAL OUTPUT | | Result from step #3 |
| RESULTS | | PASS |

**DELETE**

|  |  |  |
| --- | --- | --- |
| Scenario | Description | Pass/Fail |
| 1 | Test for correct args | PASS |
| 2 | Test for file deletion | PASS |
| 3 | Test for file existence | PASS |

**Scenario One: Test for correct args**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Type wrong number of args for command |  |
| 3 | The program should return an error message that says, “Invalid number of arguments!” |  |
| EXPECTED OUTPUT | | Error message from step #3 |
| ACTUAL OUTPUT | | Error message from step #3 |
| RESULTS | | PASS |

**Scenario Two: Test for file deletion**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Mount volume |  |
| 3 | Delete file, it should be gone from mount and catalog |  |
| EXPECTED OUTPUT | | Result from step #3 |
| ACTUAL OUTPUT | | Result from step #3 |
| RESULTS | | PASS |

**Scenario Three: Test for file existence**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Mount volume |  |
| 3 | Delete a file without a name and it should return “No file by such name to be deleted!” |  |
| EXPECTED OUTPUT | | Result from step #3 |
| ACTUAL OUTPUT | | Result from step #3 |
| RESULTS | | PASS |

**TRUNCATE (FILE)**

|  |  |  |
| --- | --- | --- |
| Scenario | Description | Pass/Fail |
| 1 | Test for correct args | PASS |
| 2 | Test for file empty | PASS |
| 3 | Test for file existence | PASS |

**Scenario One: Test for correct args**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Type wrong number of args for command |  |
| 3 | The program should return an error message that says, “Invalid number of arguments!” |  |
| EXPECTED OUTPUT | | Error message from step #3 |
| ACTUAL OUTPUT | | Error message from step #3 |
| RESULTS | | PASS |

**Scenario Two: Test for file empty**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Mount volume |  |
| 3 | Truncate file and then dump volume, there should be no data |  |
| EXPECTED OUTPUT | | Result from step #3 |
| ACTUAL OUTPUT | | Result from step #3 |
| RESULTS | | PASS |

**Scenario One: Test for file existence**

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run CMD and get to correct path |  |
| 2 | Mount volume |  |
| 3 | Truncate wrong file name and it should return “No file by such name!” |  |
| EXPECTED OUTPUT | | Result from step #3 |
| ACTUAL OUTPUT | | Result from step #3 |
| RESULTS | | PASS |

**INFO (FILE) AND SET**

These commands were not implemented due to time constraints. However, the purpose of them is to list the information of a specific file and to set a file to read only. Testing for info would include making sure the file exists and ensuring that the correct data was outputted. Testing for the set command would include ensuring that the byte for read only was set correctly and that if someone was to attempt to write to a read only file it would return an error message.