Computer Science Tripos

Part II Project Proposal Coversheet

Please fill in Part 1 of this form and attach it to the front of your Project Proposal.

Part 1

Name: Mohamed Omar

CRSID: mo501

College: Christ's College

Project Checkers:(Initials) ws390/ndl21

Title of Project: Extending littlefs to support large filesystems

Date of submission: 16/10/2023

Will Human Participants be used? No

Project Originator: Prof. Richard Mortier

Project Supervisor: Prof. Richard Mortier

Directors of Studies: Prof. Richard Mortier

Special Resource Sponsor:

Special Resource Sponsor:

Part 2

Project Checkers are to sign and comment in the students comments box on Moodle.

Part 3

For Teaching Admin use only

Date Received:

Admin Signature:

Introduction

Littlefs is a block-based filesystem that uses small, two block logs to store metadata and a copy-on-write skip list (a multi-layered linked-list) to store data blocks. It is designed for use in embedded devices. The aim of this project is to modify and extend littlefs to support large filesystems and other features required to use it with a desktop operating system.

Littlefs has built-in advantages for its intended use case compared to common filesystems (NTFS, FAT). These include bounded RAM usage even as the filesystem grows and dynamic wear levelling. Although it is not a journalling filesystem, it offers power-loss resilience thanks to strong copy-on-write guarantees on file operations.

Starting Point

I intend to use chamelon, an implementation of the majority of littlefs in OCaml, as the starting point for this project. It discards certain aspects of littlefs that are not relevant to its intended environment (MirageOS). These include a linked list which ensures that the directory tree can be traversed in constant RAM, dynamic wear levelling and tracking bad blocks.

Project Description

There are some core limitations that prevent littlefs from supporting filesystems larger than 65535 MiB.

Addressing is limited to 32-block indices, and block pointers are stored in 32 bits in littlefs. With littlefs' existing data structure for data blocks, the minimum block size for 32-bit word widths is 104 bytes, whereas for 64-bit word widths it would be 448 bytes.

Metadata pairs also incur a large storage overhead. As a result of this, the worst-case storage overhead of a file in a littlefs filesystem is a 4x increase, which occurs when it is stored as an inline file in a metadata pair. Files less than ¼ of the block size are stored as inline files.

Littlefs uses count-trailing-zeros (CTZ) skip lists to store files as there is a requirement to traverse storage in constant RAM. By using a more efficient data structure which does not satisfy this requirement, it may be possible to offset the costs of trading storage size for complexity in the other aspects of littlefs' design.

I plan to make the following modifications to littlefs to address these issues:

1. Increase limit of metadata storage usage before compaction from 50% to 75%, increasing the effective storage but doubling the cost of compaction.
2. Increase the block pointer size to 64 bits.
3. Move from using the count-trailing-zeros skip list data structure for blocks to using b-trees, forgoing the bounded RAM requirement.
4. Storing the list of used blocks in a bitmap in RAM rather than making multiple passes, again using a variable amount of RAM but reducing the runtime complexity.

In the research stage of this project, I will analyse the costs associated with these changes for operations specific to littlefs. This will allow me to make predictions about the performance of the modified filesystem, and any compromises that must be made elsewhere.

Finally, in order to mount my filesystem in user-space for testing, I will have to ensure that my changes maintain compatibility with the littlefs-fuse driver.

Testing and Evaluation

Chamelon has a set of light tests and benchmarks provided. In the research stage, I intend to add to these tests/benchmarks to cover any edge cases associated with the modifications that I will make. The primary evaluation metric will be the throughput of the filesystem, which will be measured for reads and writes in both random and sequential workloads. I will record the throughput of chamelon as-is, along with my modified version, in these tests. These performance metrics will be plotted and compared in my final evaluation, where I will determine whether the modifications to support large filesystems have incurred major performance penalties.

Success Criterion

A successful first step in this project would be an implementation that uses b-trees as its main data structure, with everything else unchanged. I.e., read, write and all other littlefs operations are supported without breaking bugs.

Then, increasing the metadata storage limit and extending the block addressing limit will achieve the primary aim of supporting large filesystems. A successful execution of this part of the project should result in a filesystem that can support sizes of at least 2TB.

However, a key aim of this project is to extend chamelon without incurring a significant performance penalty. Thus, I would consider a successful base project to incorporate all of the above while performing no more than 10% worse on average than chamelon in the tests described above. These filesystem images will be mounted in user-space using littlefs-fuse.

Extensions

Rethinking the move operation

Instead of using global state, it may be worth looking into more efficient methods of synchronising moves atomically. The directory tree could be used to find common parents for COW operations, for example. This approach is problematic if using a threaded linked-list for directory traversal as littlefs does (but this filesystem does not).

Adding PC-oriented filesystem features

Littlefs is missing features that would be expected of a pc-oriented filesystem, such as user permissions, symlinks and device files. For this extension, I will implement each of these features in turn, so that the filesystem is more suited to desktop usage.

Timetable

Date	Project Stage	Activity	Milestone
26/10/23	Research/Evaluation	Extending the test suite to support performance evaluation as well as collecting and displaying results for chameleon.	A set of graphs displaying chamelon's results in the finalised test suite.
02/11/23	Research	Investigating the performance benefits and costs of each of the 4 changes mentioned in the description.	Worked complexities for read, write, creation, deletion and move operations for a filesystem that uses b-trees for data blocks and performs compaction on the metadata pairs at 75% usage.
09/11/23	Implementation/ Write-up	Extending the block pointer address size to 64-bits and updating the rest of the codebase to reflect this.	A filesystem image with 64-bit block pointers of any size >65GB. Brief clarification of this change in a preliminary implementation chapter.
23/11/23	Implementation/ Write-up	Replacing the CTZ skip-list with a b-tree structure and updating the read operation accordingly.	Read throughput metrics for the modified filesystem (actual performance is irrelevant as long as the benchmarks and tests complete successfully). Using the analysis from the research stage, a description of the new data structure and read operation in implementation chapter.
07/12/23	BREAK	Due to the Modules of Assessment final deadline and remaining supervisions, I will likely not be able to achieve much in this block.	
21/12/23	Implementation/ Write-up	Updating the write, creation and deletion operations to support the new data structure. Raise metadata compaction limit to 75%.	Preliminary results across the entire benchmark suite for the modified filesystem.
28/12/24	BREAK	Christmas Break	
11/01/24	Implementation/ Write-up	Reimplementing the global state stored in the metadata pairs	A demonstration of the move operation working in my filesystem. Description of move

		in OCaml and restoring the atomic move operation.	operation in implementation chapter.
18/01/24	Implementation/ Write-up	Modifying the block allocator so that it stores the entire bitmap of used blocks in RAM instead of requiring multiple passes.	Running the block allocator to demonstrate that it is still functional with this method. Description of modification in implementation chapter.
25/01/24	Evaluation	I plan to write my progress report in this timeframe.	
01/02/24	Testing	Identifying and fixing any breaking bugs and remaining edge cases.	A demonstration that the filesystem passes all the tests in addition to completing all benchmarks. A 2TB image will be used for these tests.
8/02/24	Write-up	Writing the introduction chapter and structuring the dissertation	The introduction chapter completed, which will refer briefly to existing filesystems. Completed cover page, declaration of originality and proforma page outline.
15/02/24	Write-up	Writing the preparation chapter using the proposal and littlefs and chamelon documentation as a base	The preparation chapter completed (minus extensions), describing the requirements, starting point of the project, preliminary research and any algorithms/OCaml features that I have had to learn. Implementation chapter appended as-is to dissertation.
29/02/24	Write-up/Evaluation	Comparing the modified filesystem to chamelon and determining whether the base success criteria was met.	(part of the evaluation chapter) Performance comparison across the benchmarks displayed in graphs between chamelon and modified version. Commentary on performance differences compared to the predicted costs. Discussion of the trade-offs made and whether the 10% threshold was met.
07/03/24	BREAK	I may not be able to achieve much in this timeframe due to Modules of Assessment and supervision commitments.	

14/03/24	Write-up/Research	Identifying and describing a COW move algorithm that does not require global state (e.g., by looking at ZFS' approach).	(part of the preparation chapter) A description of the algorithm used for COW moves with an explanation for its correctness and showing its complexity.
28/03/24	Implementation/ Write-up	Implementing the COW tree algorithm for moves and removing global state.	A demonstration of the move operation working as intended. Description of modified move operation in implementation chapter.
04/04/24	Write-up/Evaluation	Analysing the performance benefit of removing global state.	(part of the evaluation chapter) Performance comparison across the benchmarks displayed in graphs between the filesystem with and without global state. Calculation of the performance uplift, if any. Decision on which approach is more suitable for this filesystem.
11/04/24	Implementation/ Write-up	Adding support for permissions and symlinks.	Storing a protected directory in my filesystem, containing a symlink to a text file stored elsewhere in the filesystem. Description of feature in implementation chapter.
18/04/24	Implementation/ Write-up	Adding support for device files.	A loop device created on my filesystem. Description of feature in implementation chapter.
25/05/24	Write-up/Implementation/Appendices?	Collating and editing the existing descriptions in the implementation chapter. Writing the repository overview.	A completed implementation chapter with any necessary appendices added.
02/05/24	Write-up/Conclusion	Evaluating the viability of the final filesystem for desktop usage and what, if anything, is missing. Final structuring of dissertation.	(conclusion chapter) Concluding remarks on the modified filesystem compared to littlefs and a contemporary filesystem for desktop usage. Table of contents and bibliography, if appropriate, added.

10/05/24	ONE WEEK ALLOWANCE **Submission Deadline (dissertation: 12pm, source code: 5pm)**

Resource Declaration & Backup Protocol

I will use my personal laptop to conduct work on the implementation. The specs of this device are:

CPU: 8c/16t 1.9/4.4GHz (base/boost) Ryzen 7 5800U, RAM: 16GB, SSD: 1TB, GPU: RTX 3050 Ti 4GB, OS: Ubuntu 22.04.3 LTS stored on a 128GB partition

I accept full responsibility for this machine and I have made contingency plans to protect myself against hardware and/or software failure.

Should my device fail, I will switch to working on a personal desktop PC which I shall bring to my place of residence in Cambridge.

I will use the following open-source GitHub repositories:

- littlefs-project/littlefs
- littlefs-project/littlefs-fuse
- yomimono/chamelon

I will create a private GitHub repository which I will make commits to as I work on my implementation. Separately, I will make weekly backups of the entire source code to OneDrive using the account associated with my CRSID.