Assignment4 Report

Name: Wang Jiaju - Student ID:121090544

Your report should follow the template with the following section structure.

No page limitation

Checkbox

I complete basic task correctly: ✓
I complete extra credit (if any): ✓

1 Introduction [2']

This assignment involves enhancing the file system of xv6, a simple, Unix-like teaching operating system. The main task is to implement indirect block handling to support large file management. Initially, xv6 supports only singly-indirect blocks, which are insufficient for managing large files. To address this, we expanded the file system by implementing doubly-indirect blocks, which significantly increases the maximum file size that xv6 can handle. Understanding and modifying the xv6 file system for this functionality provided deep insights into how file systems manage data structurally.

2 Design [5']

The design focuses on modifying the existing file system architecture to include doubly-indirect blocks in the inode structure. Initially, each inode in xv6 could support up to 268 blocks through 11 direct block addresses and one singly-indirect block address. The singly-indirect block can reference up to 256 additional block addresses. Our design introduces a doubly-indirect block address in place of one direct block, allowing each doubly-indirect block to reference 256 singly-indirect blocks, each of which can in turn reference 256 data blocks. This design alteration enables a file in the xv6 system to potentially consist of up to 65,803 blocks, greatly expanding the system's capacity.

Overview

The design of the enhanced file system in xv6 revolves around extending its capability to manage large files by implementing doubly-indirect blocks. This required modifying the inode data structures to accommodate additional levels of block addressing, and altering the file system's block allocation, deallocation, and access mechanisms to effectively manage these new structures. The modifications were primarily made in four key files: 'fs.c', 'file.h', 'fs.h', and 'sysfile.c'.

File Structure Modifications

"fs.h": This header file defines the structures and constants used throughout the file system. Changes to this file included updating the 'NDIRECT' constant to reflect one fewer direct block (11 instead of 12) to make room for the new doubly-indirect block. Additionally, we defined new constants for managing doubly-indirect blocks ('DNINDIRECT') and the maximum number of blocks a file can contain ('MAXFILE'), which is now significantly increased to accommodate large files.

"file.h": Modifications in this file were minimal but crucial, involving updates to the inode and file descriptor structures to support larger files and more complex block addressing. This includes adjusting inode metadata to account for the new doubly-indirect block index.

Core File System Logic

"fs.c" The core file system operations are handled in this file, where the most significant changes were implemented. Functions such as 'bmap()' and 'itrunc()' were modified to incorporate doubly-indirect blocks. In 'bmap()', logic was added to handle the allocation of blocks through doubly-indirect pointers when the file size exceeds the capacity of singly-indirect blocks. Similarly, 'itrunc()' was updated to correctly deallocate all blocks linked through doubly-indirect pointers when a file is truncated or deleted.

In the implementation of 'bmap()', when a block number beyond the reach of direct and singly-indirect blocks is requested, the function now initializes a doubly-indirect block if it does not already exist. It then navigates through the singly-indirect blocks referenced by the doubly-indirect block to locate or allocate the required block.

For 'itrunc()', additional loops were added to traverse and free the blocks at two levels of indirection, ensuring that all associated blocks are properly released when an inode's size is reset.

"sysfile.c": This file, which manages system calls related to file management, required updates to support the new file operations enabled by doubly-indirect blocks. Changes here ensured that system calls such as 'write()' and 'read()' could handle files of increased size without modification to their interfaces, providing backward compatibility with existing applications.

Testing and Validation

The modified file system was rigorously tested using the 'bigfile' command, designed to create a file that exhausts the enhanced storage capacity, thereby verifying the correct implementation of doubly-indirect blocks. Adjustments made in the testing scripts and the debugging outputs helped validate the functional expectations and performance benchmarks.

Extra Credit:

Introduction to Triple-Indirect Block Implementation

The primary challenge addressed in the extra credit task was to significantly increase the file system's maximum file size beyond what is achievable with doubly-indirect blocks. To achieve this, we implemented triple-indirect blocks. This involved extending the existing doubly-indirect architecture to include another layer of block indirection, allowing the file system to scale up and handle files of enormous sizes.

Modifications in Header and Source Files

fs_ec.h: In this header file, we extended the inode structure definitions to accommodate triple-indirect blocks. Key changes included: Introduction of a new constant "TNINDIRECT" representing the total blocks addressable by a triple-indirect block, which calculates as 256^3. Adjusting the "MAXFILE" constant to include the count from triple-indirect blocks, summing up to an extensive total that combines direct, singly-indirect, doubly-indirect, and triple-indirect blocks.

fc_ec.c: This source file saw the bulk of the algorithmic development for the triple-indirect block functionality.

Block Management Functions: Key functions such as 'bmap()' were significantly modified to support the resolution and allocation of block addresses at three levels of indirection. If a block request exceeds the capacity of doubly-indirect blocks, 'bmap()' now initiates the allocation of a triple-indirect block. It further manages the hierarchy of block allocations required to reach the desired block depth.

Inode Truncation ("itrunc()"): The function was adapted to handle the release of blocks associated with triple-indirect addressing. It now includes additional loops to traverse and free the blocks at three levels of indirection, ensuring comprehensive cleanup of disk space when files are deleted or resized.

3 Environment and Execution [2']

The modified xv6 runs in a QEMU virtual machine environment, which simulates a RISC-V processor architecture. The environment was set up using standard xv6 build tools, with modifications

made primarily in files such as ‘fs.c‘, ‘file.h‘, ‘fs.h‘, and ‘sysfile.c‘. To execute the program and test the new file system capabilities, we use the command ‘make qemu‘ to build and run the system. Verification of the large file support is done through a custom test called ‘bigfile‘, which attempts to write and manage a file significantly larger than what the original xv6 could handle. Successful execution shows the system writing up to 65,803 blocks, confirming that the new file handling capabilities are functioning correctly.

4 Conclusion [2’]

This assignment provided valuable experience in manipulating and understanding lower-level system functionalities, particularly in file system management. By extending the xv6 file system to support large files through doubly-indirect blocks, not only was the theoretical knowledge reinforced, but practical, hands-on experience in system programming was also gained. The challenge of integrating new components into an existing system architecture taught problem-solving skills and the importance of detailed system design.