Assignment Report: Large File and Sym-links Support in xv6 File System

Chaoren WANG - 122090513

1 Introduction [2']

This assignment involves enhancing the file system of 'xv6' by introducing support for doubly indirect blocks and symbolic links. The aim is to expand the file system's capabilities, allowing larger file sizes and flexible file path references. My work included implementing these features in the inode structure, block allocation routines, and modify system calls like 'open' and 'symlink'.

2 Design [5']

2.1 Doubly Indirect Block Implementation

To support larger files, I updated the inode structure to include a pointer for doubly indirect blocks. Specifically, the pointer to sub inode is saved in the 'addr[]'. The 'bmap' function was modified to handle doubly indirect block allocation and traversal. Additionally, the 'itrunc' function was updated to properly free the doubly indirect blocks during inode truncation.

The constants in 'fs.h' were adjusted: 1) 'NDIRECT' reduced to 11, 2) Added 'DNINDIRECT' to represent the doubly indirect block count, 3) Updated 'MAXFILE' to reflect the expanded maximum file size.

2.2 Symbolic Link System Calls

Two system calls were implemented: 1) 'sys_symlink': Creates a symbolic link by writing the target path into the data block of a new inode marked as 'T_SYMLINK'. 2) 'sys_open': Recursively resolves symbolic links unless the 'O_NOFOLLOW' flag is set. It also detects and prevents cyclic links using a retry threshold (maximum cyclic threshold set to 10).

Modifications to the 'open' system call ensure compatibility with existing and new file types, handling edge cases like invalid paths and cyclic links.

2.3 Updated System Constants and Structures

The inode structure ('inode' and 'dinode') was modified to accommodate doubly indirect blocks, with an proper length 'addrs' array.

3 Environment and Execution [2']

3.1 Environment

The project was developed using the provided 'xv6' virtual machine ('CSC3150_xv6.qcow2'). Figure 1 shows the virtual machine setup.

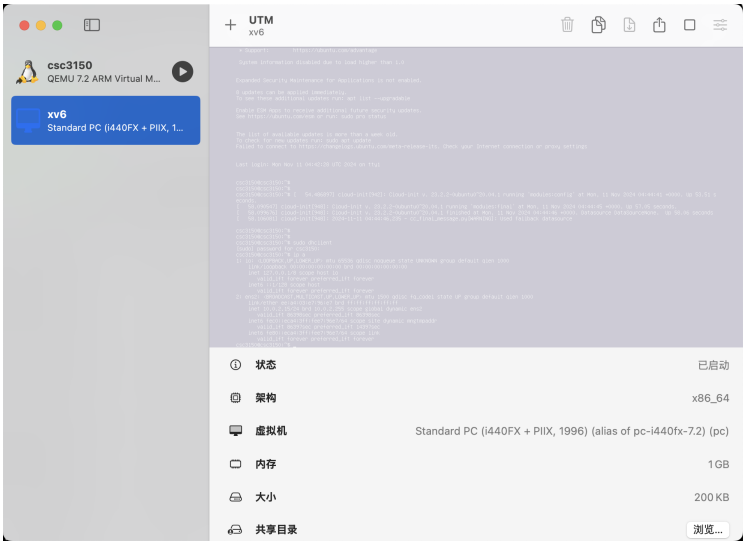

Figure 1: Running ‘xv6’ in the UTM environment.

3.2 Execution

The system was compiled and executed using ‘make qemu’. Functional testing involved: 1) Verifying file operations with larger files to test doubly indirect block functionality by running ‘bigfile’. 2) Using symbolic link tests (‘symlinktest’) under user directory to confirm the correct handling of link creation, traversal, and cyclic detection.

Test results in Figure 2 and Figure 3 confirm the correct implementation of both features.

Figure 2: Successful test result after running ‘bigfile’ in xv6 system

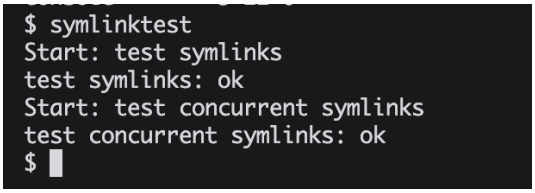

Figure 3: Successful test result after running ‘symlinktest’ in xv6 system

4 Conclusion [2’]

This assignment deepened my understanding of file system structures and operations. Implementing doubly indirect blocks reinforced concepts of hierarchical block allocation, while symbolic links highlighted the importance of flexible file path resolution and error handling. These enhancements align the ‘xv6’ file system with features found in modern operating systems, providing valuable insights into scalable file system design and management.