Bilkent University

Department of Computer Engineering

2022 - 2023 Fall Semester

CS 342 Operating Systems

Project #4- Linux ext2 File System

Authors:

Efe Erkan	21902248
Recep Uysal	21803637

Instructor: İbrahim Körpeoğlu

Implementation

In this project, we implemented a C program called *diskprint* which takes an ext2 formatted soft disk and parses the information about the disk. The information given by the program is the superblock information, the content of the root directory (file names in the root directory) and inode information of each regular file in the root directory.

In order to read blocks from the disk, *lseek()* and *read()* system calls were used. After reading the blocks, fields of superblock, group descriptor table, inodes and directory entries are taken by using the ext2 file system documentation[1]. In order to achieve this, two important functions which are *retrieve_field()* and *retrieve_field_str()* were implemented. These functions take a block from the disk, an offset value and size value as input. First of all, these functions flip the information in the block because the disk system uses Little Endian addressing and after that they convert binary numbers into integers (retrieve_field) or strings (retrieve_field_str). The offset and size information of each field are taken from the ext2 file system documentation.

Experiments

First experiment of this project is measuring the time required to format an ext2 soft disk with various block counts and block sizes. A soft disk is created with **dd** command and formatted with **mke2fs** command in Linux. An example of usage of these command can be:

```
dd if=/dev/zero of=disk1 bs=4K count=1000

/sbin/mke2fs -t ext2 -r 0 -b 4096 -I 128 disk1 1000
```

These commands create a soft disk named disk1 which has 1000 blocks and each block size is 4KB. In this experiment, we created different soft disks and measured the time for formatting by using **time** command. An example invocation can be:

```
time /sbin/mke2fs -t ext2 -r 0 -b 4096 -I 128 disk3 3000
```

[1] https://www.nongnu.org/ext2-doc/ext2.html

First of all, the block size was kept same in 4KB and block counts changed:

Block Count (Block Size is 4KB)	Time (ms)
1000	192
2000	192
3000	202
4000	598
5000	613
6000	682

By observing the results, it can be said that when we increase the number of blocks, the time also increases almost linearly. If we look closely, there is a huge jump between block counts 3000 and 4000. The reason for that might be disk caching because 4000 blocks might not be enough for one caching operation.

Second of all, the block size was kept same in 1000 blocks and block sizes changed:

Block Size (Block Count is 1000)	Time (ms)
1 KB	162
2 KB	278
4 KB	267
8 KB	299
16 KB	314
32 KB	307

Observing the results, it can be said that when we increase the block size, the time also increases but compared with the previous experiment, the rate of this increase is not linear because the execution time is not doubled if the block size is doubled. Compared with the previous experiment, it can be said that 1000 blocks are cached by the system and because of that, the execution time does not rise rapidly.

After observing the formatting on different block sizes and count, we did another experiment by using *diskprint* program for the required time to parse a disk, in other words getting superblock information. In order to achieve this, we used the disks in the first experiment which have the same block size (4KB) and different block counts (1000 to 6000). For the sake of correctness of this experiment, the content of the root directories are the same and all root directories contain these files: file1.txt, file2.txt, file3.bin and file4.txt.

Block Count (Block Size is 4 KB)	Parse Time (ms)
1000	263
2000	313
3000	297
4000	308
5000	300
6000	284

Looking at the results of this experiment, we can state that the time required to parse disk does not change much with different block counts. If we analyze our approach to implement the parsing, it makes sense that parse time is not dependent on the block count because the program always reaches certain blocks such as superblock, group descriptor table, inode table and directory entries.