CS 342: Operating Systems
Project 4 Report

Yusuf Avci - 21702724

Section 1

Table Of Contents

Introduction

Source codes are in the appendix to make the report more readable.

Part 1

Created a file 200MB in size by setting N (block count) to 50000.

P1 Output:
```
$ ./p1 50000
Created file with name: p1_file
Writing 50000 blocks.
Done Writing
```

Used this command to *format* the file. The b option is used to specify the block size.
```
$ mkfs.ext4 -b 4096 p1_file
```

Output:
```
mke2fs 1.44.1 (24-Mar-2018)
Discarding device blocks: done
Creating filesystem with 50000 4k blocks and 50048 inodes
Filesystem UUID: cb789409-7aa0-4b83-b8c3-277db5ffebc8
Superblock backups stored on blocks:
    32768

Allocating group tables: done
Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done
```

50048 Inodes are generated.

Created a directory named mounpoint and used below command to mount.
```
$ sudo mount -o loop p1_file ./mountpoint
```

After mounting, there was a lost+found named directory in the mount directory.

Created 3 files with

```
$ sudo touch file1
$ sudo touch file2
$ sudo touch file3
```

commands.

Mount point content after creating the files.

```
$ ls
file1  file2  file3  lost+found
```

Changed back to the parent directory and unmounted via

```
$ sudo umount mountpoint/
```

Then, changed to mountpoint. Mountpoint was empty after umount. After going to parent directory again and mounting again with

```
$ sudo mount -o loop p1_file ./mountpoint
```

As a result, the files returned.

```
$ ls
file1  file2  file3  lost+found
```

Used dumpe2fs to see the dump.

```
$ sudo dumpe2fs p1_file
dumpe2fs 1.44.1 (24-Mar-2018)
Filesystem volume name:   <none>
Last mounted on:
/home/yusuf/CS/CS342/OSProjects/Project4/mountpoint
Filesystem UUID:          cb789409-7aa0-4b83-b8c3-277db5ffebc8
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      has_journal ext_attr resize_inode
dir_index filetype extent 64bit flex_bg sparse_super large_file
huge_file dir_nlink extra_isize metadata_csum
Filesystem flags:         signed_directory_hash
Default mount options:    user_xattr acl
Filesystem state:         clean
Errors behavior:          Continue
Filesystem OS type:       Linux
Inode count:              50048
Block count:              50000
Reserved block count:     2500
Free blocks:              44278
Free inodes:              50034
First block:              0
Block size:               4096
```

```
Fragment size:            4096
Group descriptor size:    64
Reserved GDT blocks:      24
Blocks per group:         32768
Fragments per group:      32768
Inodes per group:         25024
Inode blocks per group:   782
Flex block group size:    16
Filesystem created:       Mon May 25 02:30:40 2020
Last mount time:          Mon May 25 02:41:16 2020
Last write time:          Mon May 25 02:43:26 2020
Mount count:              2
Maximum mount count:      -1
Last checked:             Mon May 25 02:30:40 2020
Check interval:           0 (<none>)
Lifetime writes:          257 kB
Reserved blocks uid:      0 (user root)
Reserved blocks gid:      0 (group root)
First inode:              11
Inode size:          128
Journal inode:            8
Default directory hash:   half_md4
Directory Hash Seed:      34d3cc04-974f-4cee-868a-1cc635999934
Journal backup:           inode blocks
Checksum type:            crc32c
Checksum:                 0xd1c5251d
Journal features:         journal_64bit journal_checksum_v3
Journal size:             16M
Journal length:           4096
Journal sequence:         0x00000007
Journal start:            0
Journal checksum type:    crc32c
Journal checksum:         0x689056c3


Group 0: (Blocks 0-32767) csum 0xcb26 [ITABLE_ZEROED]
  Primary superblock at 0, Group descriptors at 1-1
  Reserved GDT blocks at 2-25
  Block bitmap at 26 (+26), csum 0x7612eddf
  Inode bitmap at 28 (+28), csum 0xf0626d0f
  Inode table at 30-811 (+30)
  27072 free blocks, 25010 free inodes, 2 directories, 25010 unused
inodes
```

```
  Free blocks: 5696-32767
  Free inodes: 15-25024
Group 1: (Blocks 32768-49999) csum 0x6e4c [INODE_UNINIT,
ITABLE_ZEROED]
  Backup superblock at 32768, Group descriptors at 32769-32769
  Reserved GDT blocks at 32770-32793
  Block bitmap at 27 (bg #0 + 27), csum 0xc2420dbb
  Inode bitmap at 29 (bg #0 + 29), csum 0x00000000
  Inode table at 812-1593 (bg #0 + 812)
  17206 free blocks, 25024 free inodes, 0 directories, 25024 unused
inodes
  Free blocks: 32794-49999
  Free inodes: 25025-50048
```

- 27072+17206 = 44278 blocks are free.
- There are 2 groups since there are 50000 blocks (200 MB) and each group has 32K blocks.
- From “Block bitmap at 26 (+26), csum 0x7612eddf” I can see that bitmap is in block 26.
- Bitmap is big enough because it uses a block which is 4096B which means (4 * 8) 32K bits and 32K bits are (just) enough to map 32K blocks.
- Inode bitmap is at block 28.
- 1 block is occupied by inode bitmap because the number of inodes is lower than 32K in group 0.
- Inode table is at blocks 30-811. 811-30+1 = 782 blocks are occupied by the inode table.
- 27072 blocks are free in group 0.

Part 2

Number of blocks is not always Size / 512 because of the holes.
P2 Output:

```
./p2 .
Given directory path: .
Added '/' to the end: ./
Current file path: ./p3
File (or subdirectory) name: p3
Inode number: 800615
File type: Regular file
```

```
Number of Blocks: 40
Size (In Bytes): 17112
User ID: 1000

Current file path: ./p1
File (or subdirectory) name: p1
Inode number: 787511
File type: Regular file
Number of Blocks: 24
Size (In Bytes): 11568
User ID: 1000

Current file path: ./p3.c
File (or subdirectory) name: p3.c
Inode number: 816864
File type: Regular file
Number of Blocks: 8
Size (In Bytes): 2323
User ID: 1000

Current file path: ./p2.c
File (or subdirectory) name: p2.c
Inode number: 816857
File type: Regular file
Number of Blocks: 8
Size (In Bytes): 2817
User ID: 1000

Current file path: ./..
File (or subdirectory) name: ..
Inode number: 403556
File type: Directory
Number of Blocks: 8
Size (In Bytes): 4096
User ID: 1000

Current file path: ./p2
File (or subdirectory) name: p2
Inode number: 800614
File type: Regular file
Number of Blocks: 40
Size (In Bytes): 16496
User ID: 1000
```

```
Current file path: ./.
File (or subdirectory) name: .
Inode number: 816848
File type: Directory
Number of Blocks: 8
Size (In Bytes): 4096
User ID: 1000

Current file path: ./notes
File (or subdirectory) name: notes
Inode number: 800613
File type: Regular file
Number of Blocks: 8
Size (In Bytes): 435
User ID: 1000

Current file path: ./p1.c
File (or subdirectory) name: p1.c
Inode number: 816854
File type: Regular file
Number of Blocks: 8
Size (In Bytes): 1114
User ID: 1000

Current file path: ./mountpoint
File (or subdirectory) name: mountpoint
Inode number: 816852
File type: Directory
Number of Blocks: 8
Size (In Bytes): 4096
User ID: 1000

Current file path: ./p1_file
File (or subdirectory) name: p1_file
Inode number: 800616
File type: Regular file
Number of Blocks: 312
Size (In Bytes): 204800000
User ID: 1000

Current file path: ./ydk
File (or subdirectory) name: ydk
```

```
Inode number: 917524
File type: Directory
Number of Blocks: 8
Size (In Bytes): 4096
User ID: 1000

Current file path: ./Makefile
File (or subdirectory) name: Makefile
Inode number: 787380
File type: Regular file
Number of Blocks: 8
Size (In Bytes): 239
User ID: 1000

Current file path: ./myclib.c
File (or subdirectory) name: myclib.c
Inode number: 816850
File type: Regular file
Number of Blocks: 8
Size (In Bytes): 689
User ID: 1000
```

Part 3

Created a 400 MB size with p1 and did 900 random reads with K = 100 and outputted each one's result. As this would be too large omitted most of the reads (No further change is done to the output only unnecessarily long parts are removed).

First Output:

```
File size: 409600000 B
Starting Random Access
Read Count: 0
Read From Index: 264146302
Read Time: 9 microseconds
Read:
0000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000

Read Count: 1
Read From Index: 364738943
Read Time: 4 microseconds
Read:
000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000

Read Count: 2
Read From Index: 97380723
Read Time: 5 microseconds
Read:
000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000

Read Count: 3
Read From Index: 12741452
Read Time: 4 microseconds
Read:
000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000

...

Read Count: 897
Read From Index: 116758226
Read Time: 1 microseconds
Read:
000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000

Read Count: 898
Read From Index: 51594921
Read Time: 1 microseconds
Read:
000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000

Read Count: 899
Read From Index: 389466801
Read Time: 1 microseconds
Read:
000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000
```

```
Random Access Done
Total elapsed time: 1510 microseconds
Average access time: 1 microseconds
```

Random access happened in 1 microseconds on average.
Rebooted the computer and tried again.

Second Output:

```
File size: 409600000 B
Starting Random Access
Read Count: 0
Read From Index: 123881831
Read Time: 125 microseconds
Read:
000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000

Read Count: 1
Read From Index: 148896333
Read Time: 103 microseconds
Read:
000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000

...

Read Count: 898
Read From Index: 384196015
Read Time: 90 microseconds
Read:
000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000

Read Count: 899
Read From Index: 344669742
Read Time: 3 microseconds
Read:
000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000

Random Access Done
Total elapsed time: 78139 microseconds
```

```
Average access time: 86 microseconds
```

This time it is extremely slower. 86 microseconds.

Dropped the cache with `sudo echo 3 > /proc/sys/vm/drop_caches` command. However, I had to write the `sudo su` command beforehand because it gave permission denied error without that. My interpretation is that the file is put into main memory when created. That's why access is very fast in the first part. However, in the second part, read is from the disk.

Third Output:

```
File size: 409600000 B
Starting Random Access
Read Count: 0
Read From Index: 118368430
Read Time: 241 microseconds
Read:
000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000

Read Count: 1
Read From Index: 276480125
Read Time: 106 microseconds
Read:
000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000

...

Read Count: 898
Read From Index: 407562895
Read Time: 83 microseconds
Read:
000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000

Read Count: 899
Read From Index: 132532983
Read Time: 83 microseconds
Read:
000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000
```

```
Random Access Done
Total elapsed time: 85927 microseconds
Average access time: 95 microseconds
```

Average time is 95 microseconds this time. It is even slower. However, not much slower than the second try. This slowness may be by chance because the computer didn’t have any info in the cache after the reboot as well. Or if any info was in the cache, it may deleted and speed is decreased slightly.

Appendix

Part 1 (p1.c)

```
#define BLOCK_SIZE 4096
#define FILE_NAME "p1_file"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>

void write_block(int fd);

int main(int argc, char *argv[]) {

	// Argument count control
	if (argc != 2) {
	printf ("Wrong number of arguments entered!\n");
	return 1;
	}

	// Argument validity control
	int N = atoi(argv[1]);
	if (N <= 0) {
	printf("%s\n", "Please gave a positive integer as block
number.");
	return 1;
	}
```

```
    // File creation
    printf("%s%s\n", "Created file with name: ", FILE_NAME);
    int fd = open(FILE_NAME, O_RDWR | O_CREAT | O_TRUNC, 0600);
    if (fd < 0) {
    printf("%s\n", "Could not create the file.");
    return 1;
    }

    // Writing
    printf("%s%d%s\n", "Writing ", N, " blocks.");
    for( int i = 0; i < N; i++) {
    write_block(fd);
    }

    // Closing
    printf("%s\n", "Done Writing");
    close(fd);
    return 0;
}

void write_block(int fd) {

    size_t length = BLOCK_SIZE;
    char str[BLOCK_SIZE];
    char *dest = &str[0];

    while (length-- > 0) {
    *dest++ = '0';
    }
    *dest = '\0';

    write(fd, str, BLOCK_SIZE);
}
```

Part 2 (p2.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <sys/stat.h>
```

```
int main (int argc, char *argv[]) {
        struct dirent *pDirent;
        struct stat stats;
        DIR *pDir;

        // Argument count control
        if (argc != 2) {
        printf ("Wrong number of arguments entered!\n");
        return 1;
        }

        // Opening the directory
        pDir = opendir (argv[1]);
        if (pDir == NULL) {
        printf ("Cannot open directory '%s'\n", argv[1]);
        return 1;
        }

        // Adding '/' to end of the path if there is no '/' in the end
of path
        char dir_path[256];
        strcpy(dir_path, argv[1]);
        printf("%s%s\n", "Given directory path: ", dir_path);
        if(dir_path[strlen(dir_path) - 1] != '/') {
        strcat(dir_path, "/");
        printf("%s%s\n", "Added '/' to the end: ", dir_path);
        }
        printf("\n");

        // Iterating every directory entry in the directory
        char cur_file_path[256];
        while ((pDirent = readdir(pDir)) != NULL) {

        // Finding paths of each dirent
        strcpy(cur_file_path, dir_path);
        strcat(cur_file_path, pDirent->d_name);
        printf("%s%s\n", "Current file path: ", cur_file_path);

        // Print required properties
        printf ("%s%s\n", "File (or subdirectory) name:
",pDirent->d_name);
        if (stat(cur_file_path, &stats) == 0) {
                printf("%s%ld\n", "Inode number: ", (long) stats.st_ino);
```

```
          printf("%s", "File type: ");

          switch (stats.st_mode & S_IFMT) {
               case S_IFBLK:  printf("%s", "Block device\n");
break;
               case S_IFCHR:  printf("%s", "Character device\n");
break;
               case S_IFDIR:  printf("%s", "Directory\n");
break;
               case S_IFIFO:  printf("%s", "FIFO/pipe\n");
break;
               case S_IFLNK:  printf("%s", "Symlink\n");
break;
               case S_IFREG:  printf("%s", "Regular file\n");
break;
               case S_IFSOCK: printf("%s", "Socket\n");
break;
               default:        printf("%s", "Unknown?\n");
break;
          }

          printf("%s%lld\n", "Number of Blocks: ", (long long)
stats.st_blocks);
          printf("%s%lld\n", "Size (In Bytes): ",(long long)
stats.st_size);
          printf("%s%ld\n\n", "User ID: ", (long) stats.st_uid);
     }
     else {
          printf("Unable to get file properties.\n");
          printf("Please check whether '%s' file exists.\n\n",
pDirent->d_name);
     }
     }

     // Close
     closedir (pDir);
     return 0;
}
```

Part 3 (p3.c)

```
#define RANDOM_ACCESS_COUNT 900
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <time.h>
#include <sys/time.h>
#include <sys/stat.h>

unsigned long getElapsedTime(struct timeval *start, struct timeval
*end);

int main (int argc, char *argv[]) {
      srand((unsigned int)(time(NULL)));
      struct timeval start, end;

      // Argument count control
      if (argc != 3) {
      printf ("Wrong number of arguments entered!\n");
      return 1;
      }

      // K validity check
      int K = atoi(argv[1]);
      if (K <= 0) {
      printf("%s\n", "Please gave a positive integer for K");
      return 1;
      }

      // Opening the file
      char *F = argv[2];
      int fd = open(F, O_RDONLY);
      if (fd < 0) {
      printf("%s%s\n", "Could not find (or open) a file named: ", F);
      return 1;
      }

      size_t file_size;
      struct stat file_info;
      if (stat(F ,&file_info) == 0) {
      file_size = (size_t)file_info.st_size;
      }
      else {
```

```
    printf("%s\n", "Couldn't determine the file size.");
    return 1;
    }
    printf("%s%ld%s\n", "File size: ", file_size, " B");

    if(K > file_size) {
    printf("%s\n", "K bigger than file size");
    return 1;
    }
    // Make random accesses
    unsigned long elapsed_time = 0;
    unsigned long read_time;
    char buffer[K];
    printf("%s\n", "Starting Random Access");
    for(int read_count = 0; read_count < RANDOM_ACCESS_COUNT;
read_count++) {

    // Get a random index
    size_t index = (double) rand() / RAND_MAX * (file_size - K +
1);

    // Set fd
    lseek(fd, index, SEEK_SET);

    // Calculate read time
    gettimeofday(&start, NULL);
    read(fd, buffer, K);
    gettimeofday(&end, NULL);
    read_time = getElapsedTime(&start, &end);
    elapsed_time += read_time;
    printf("%s%d\n", "Read Count: ", read_count);
    printf("%s%ld\n", "Read From Index: ", index);
    printf("%s%ld%s\n", "Read Time: ", read_time, " microseconds");
    printf("%s\n", "Read: ");
    for(int j = 0; j < K; j++) {
        printf("%c", buffer[j]);
    }
    printf("\n\n");
    }
    printf("%s\n", "Random Access Done");

    unsigned long average_time = elapsed_time /
RANDOM_ACCESS_COUNT;
```

```
	printf("%s%ld%s\n", "Total elapsed time: ",elapsed_time, " 
microseconds");
	printf("%s%ld%s\n", "Average access time: ",average_time, " 
microseconds");

	close(fd);
	return 0;
}

unsigned long getElapsedTime(struct timeval *start, struct timeval 
*end) {
	return ((end->tv_sec - start->tv_sec) * 1000000) + 
(end->tv_usec - start->tv_usec);
}
```