CS 342

Operating Systems

Project 4

Mian Usman Naeem Kakakhel

21701015

Sec: 01

26/05/2020

Part 1)
File vDisk was created with command:
./p1 20000
Creating a file “vDisk” with the size 8.19 MB.

Then Using the command:
mkfs.ext4 -c vDisk -b 4096
The output was:
mke2fs 1.44.1 (24-Mar-2018)
Creating filesystem with 20000 4k blocks and 20000 inodes

Checking for bad blocks (read-only test): 0.00% done, 0:00 elapsed. (0/0/0 errdone
Allocating group tables: done
Writing inode tables: done
Creating journal (1024 blocks): done
Writing superblocks and filesystem accounting information: done

As can be seen, the number of inodes created are 20000.

When I mounted the disk, I created a text file of 20 kb in the disk and then when I unmount the disk, I could not see the file over there. But then when I mounted it again, I could see the file that I had created. Finally the disk was unmount again.

Then the command was run:
Sudo dumpe2fs vDisk

Output was generated:
dumpe2fs 1.44.1 (24-Mar-2018)
Filesystem volume name: <none>
Last mounted on: <not available>
Filesystem UUID: bc532fe7-9744-487c-b607-45b3f1d3e0f8
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: 20000
Block count: 20000
Reserved block count: 1000

Free blocks: *18331*
Free inodes: *19988*
First block: *0*
Block size: *4096*
Fragment size: *4096*
Group descriptor size: *64*
Reserved GDT blocks: *9*
Blocks per group: *32768*
Fragments per group: *32768*
Inodes per group: *20000*
Inode blocks per group: *625*
Flex block group size: *16*
Filesystem created: *Mon May 25 23:58:18 2020*
Last mount time: *Tue May 26 00:13:09 2020*
Last write time: *Tue May 26 00:13:33 2020*
Mount count: *4*
Maximum mount count: *-1*
Last checked: *Mon May 25 23:58:18 2020*
Check interval: *0 (<none>)*
Lifetime writes: *9 MB*
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: *5c7f35da-e477-4282-8786-36e88d23ffbc*
Journal backup: *inode blocks*
Checksum type: *crc32c*
Checksum: *0xe969d68f*
Journal features: *journal_64bit journal_checksum_v3*
Journal size: *4096k*
Journal length: *1024*
Journal sequence: *0x00000009*
Journal start: *0*
Journal checksum type: *crc32c*
Journal checksum: *0xe9ca80d2*

Group 0: (Blocks 0-19999) csum 0xca72 [ITABLE_ZEROED]
Primary superblock at 0, Group descriptors at 1-1
Reserved GDT blocks at 2-10
Block bitmap at 11 (+11), csum 0x0cad3c7e

Inode bitmap at 27 (+27), csum 0x2d170988
Inode table at 43-667 (+43)
18331 free blocks, 19988 free inodes, 2 directories, 19987 unused inodes
Free blocks: 1669-19999
Free inodes: 12, 14-20000

Since my disk size is 8.9 MB, I have 20K blocks in my disk. Since my blocks can be accommodated in 1 group, I only have group 0. The block bitmap is on 11 block. Since there are 4096 bytes in a block, and there are 32K Blocks in a Group, one block has bits 4096 * 8 = 32K bits. Thus block bitmap can contain info about all of the blocks in the group. Inode bitmap is located at 27 block. The inode bitmap has 4096 B size. Inode table is at 43 block. Inode table is from 43-667 block. There are 19988 free inodes.

PART 2)

Output of part2:

Name: .
User ID: 1000
Inode Number: 253
File Type: Directory
Number Of Blocks: 8
Size in Bytes: 4096

Name: ..
User ID: 1000
Inode Number: 1075281
File Type: Directory
Number Of Blocks: 8
Size in Bytes: 4096

Name: bin
User ID: 1000
Inode Number: 13480
File Type: File
Number Of Blocks: 16400
Size in Bytes: 8394210

Name: Makefile
User ID: 1000
Inode Number: 256
File Type: File
Number Of Blocks: 1

Size in Bytes: 135

Name: mDir
User ID: 1000
Inode Number: 257
File Type: Directory
Number Of Blocks: 0
Size in Bytes: 0

Name: outputs.txt
User ID: 1000
Inode Number: 1156190
File Type: File
Number Of Blocks: 8
Size in Bytes: 2899

Name: p1
User ID: 1000
Inode Number: 627745
File Type: File
Number Of Blocks: 24
Size in Bytes: 11328

Name: p1.c
User ID: 1000
Inode Number: 255
File Type: File
Number Of Blocks: 8
Size in Bytes: 750

Name: p2
User ID: 1000
Inode Number: 13333
File Type: File
Number Of Blocks: 40
Size in Bytes: 16800

Name: p2.c
User ID: 1000
Inode Number: 258
File Type: File
Number Of Blocks: 8
Size in Bytes: 2059

Name: p3
User ID: 1000
Inode Number: 264
File Type: File
Number Of Blocks: 40
Size in Bytes: 17152

Name: p3.c
User ID: 1000
Inode Number: 12772
File Type: File
Number Of Blocks: 8
Size in Bytes: 1316

Name: project4.pdf
User ID: 1000
Inode Number: 254
File Type: File
Number Of Blocks: 256
Size in Bytes: 128542

Name: vDisk
User ID: 1000
Inode Number: 13831
File Type: File
Number Of Blocks: 160000
Size in Bytes: 81920000

PART 3)
An 8.3 MB “bin”, file was created and it was given to the program with the the command:
./p3 400000 bin

When it was first run, the average time given was 1058 milli-seconds, but after that I got 127 milli-seconds seconds. When I restarted the computer, I got 1220 milli-seconds after that I got 139 Milli-seconds and finally after clearing the cache I got 995 milli-seconds but after that I got 140 milli-seconds. Now this first time the time is way greater because the data is present in the cache. But when the data enters the cache, the file can easily be read from the cache.

SOURCE CODES:

PART1)

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


int main(int argc, char** argv){
   //ensure that the arguments are given correctly
   if (argc != 2){
       printf ("number of arguments given are not accurate. Please try
again.\n");
       exit(0);
   }
   int N = atoi(argv[1]);

   int fd = open("vDisk", O_CREAT|O_WRONLY|O_TRUNC);
   if (fd < 0){
       printf ("File could not be created.\n");
       exit(0);
   }

   for (int i = 0; i < N; i++){
       for (int j = 0; j < 512; j++){
           if (write(fd, "dilwhich", 8) != 8){
               printf("Failed to write into file.\n");
           }
       }
   }

   close(fd);
   return 0;
}
```

PART2)

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <dirent.h>
#include <string.h>


int main(int argc, char** argv){
   //ensure that the arguments are given correctly
   if (argc != 2){
       printf ("number of arguments given are not accurate. Please try
again.\n");
       exit(0);
   }
   char* address = argv[1];
   if (address[strlen(address) - 1] != '/'){
       strcat(address, "/");
   }
   DIR *pDir;
   struct dirent *pDirent;
   struct stat fileStat;
   pDir = opendir (address);
   if (!pDir){
       printf ("Wrong Directory address. Please try again.\n");
       exit(0);
   }

   while ((pDirent = readdir(pDir)) != NULL) {
       int addressLen = strlen(address);
       char tempAdd[addressLen + 1];
       strcpy(tempAdd, address);
       strcat(tempAdd, pDirent->d_name);
       printf ("-----------------------------\n");
```

```
        stat(tempAdd,&fileStat);
        printf("Name: \t\t\t%s\n", pDirent->d_name);
        printf("User ID: \t\t%d\n", fileStat.st_uid);
        printf("Inode Number: \t\t%ld\n", fileStat.st_ino);
        if (S_ISREG(fileStat.st_mode)) {
            printf("File Type: \t\tFile\n");
        }
        else if (S_ISDIR(fileStat.st_mode)) {
            printf("File Type: \t\tDirectory\n");
        }
        else if (S_ISCHR(fileStat.st_mode)) {
            printf("File Type: \t\tCharacter Device\n");
        }
        else if (S_ISBLK(fileStat.st_mode)) {
            printf("File Type: \t\tBlock Device\n");
        }
        else if (S_ISFIFO(fileStat.st_mode)) {
            printf("File Type: \t\tFIFO\n");
        }
        else if (S_ISLNK(fileStat.st_mode)) {
            printf("File Type: \t\tSymbolic Link\n");
        }
        else if (S_ISSOCK(fileStat.st_mode)) {
            printf("File Type: \t\tSocket\n");
        }
        printf("Number Of Blocks: \t%ld\n", fileStat.st_blocks);
        printf("Size in Bytes: \t\t%ld\n", fileStat.st_size);
    }

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

PART3)

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


int main(int argc, char** argv){
   time_t t;
   srand((unsigned) time(&t));
   //ensure that the arguments are given correctly
   if (argc != 3){
       printf ("number of arguments given are not accurate. Please try
again.\n");
       exit(0);
   }
   int K = atoi(argv[1]);
   int fd = open(argv[2], O_RDONLY);
   if (fd < 0){
       printf ("File could not be opened.\n");
       exit(0);
   }

   lseek(fd, 0, SEEK_END);
   struct stat fileStat;
   stat(argv[2],&fileStat);
   int size = fileStat.st_size;
   lseek(fd, 0, SEEK_SET);

   int range = size - K;
   if (range < 1){
       printf ("K greater than file Size, try again.\n");
   }
```

```
    struct timeval tv;
    suseconds_t totaltime = 0;
    for (int i = 0; i < 200; i++){
        gettimeofday(&tv, NULL);
        suseconds_t starttime = tv.tv_usec;
        lseek(fd, (int)(rand()%range), SEEK_SET);
        char data[K];
        read(fd, data, K);
        gettimeofday(&tv, NULL);
        totaltime = totaltime + (tv.tv_usec - starttime);
    }
    totaltime = totaltime/200;
    printf("Time taken: %ld micro Seconds\n", totaltime);

    close(fd);
    return 0;
}
```