CS342 Operating Systems – Spring 2020
Project 4: Linux File System Experiments

Assigned: May 16, 2020
Due date: May 27, 2020. Time: 23:55 *Document Version*: 1.1

*You will do this project **individually**. You have to program in C and Linux. We will do our tests in the following distribution of Linux: **Ubuntu 18 LTS – 64 bit**.*

In this project you will write several simple programs and do some simple experiments. You will put everything, the programs (source codes), their various outputs, the outputs of utilities, and your descriptions and interpretations into a report (i.e. document them) that will be submitted as a **pdf** file. You will also upload the program sources and a Makefile. We should be able to just type make and obtain the executable files. All these files (pdf report, README, Makefile, program sources) will be put into a folder, tarred and gzipped and submitted as a single file in Moodle.

1) Write a simple **program p1** that will create a file and will put into the file N blocks of information where each block is of size 4094 bytes. Hence file size will be N * 4096 bytes. N is a parameter to the program. We will run the program as p1 N. The information to write to the file can be all zeros. In your program, you will use open, read, write, close system calls (Linux low-level I/O functions). In this program you will just use the write system call, not the read system call.

Run the program to **generate a file** that is large enough (10s or 100s of MB). It will act as a disk. i.e., it will be a **virtual disk**.

Then **format** the virtual disk with Linux **ext4** file system (i.e., create an ext4 file system on the virtual disk). You can do this by **mkfs.ext4** command in Linux. You will use 4096 (bytes) as the block size. See the output of mkfs command. How many inodes are generated? In Linux file system, inodes are put into an inode table that occupies some number of contiguous disk blocks (portions of the table can be in different groups of blocks). There is also an inode bitmap that shows which inodes in the inode table are used and which inodes are free.

Then, **mount** the ext4 file system in the virtual disk using the mount command to an empty subdirectory (let us call this as your *mount directory*). Then change to that mount directory. Create some files there. Then unmount the virtual disk (use umount command). Change to the mount directory again. Are you seeing the files? Now mount again. Are you seeing the files now in the mount directory? Now unmount again.

Using the **dumpe2fs** command, dump information about the ext4 file system in the virtual disk. See the output. How many blocks are free? Ext4 file system divides the blocks of the disk into groups. Each group has 32K blocks. How many groups are there? In group 0, where is the bitmap (on which block)? Is the bitmap big enough to contain information about all blocks in that group? Where is the inode bitmap (indicates if an inode in the inode table is used or not)? How many blocks are

occupied by the inode bitmap? Where is the inode table in group 0? How many blocks are occupied by the inode table? How many blocks are free in group 0?

2) Write a **program** p2 that will list all files and subdirectories in a given directory. We will run the program as: p2 P. P is the pathname of the directory. Your program will use **opendir** and **readdir** functions to read the *directory entries* of the given directory. For each directory entry, which can correspond to a file or a subdirectory, you will use the **stat** function to learn more about the attributes of the related file or subdirectory. For each file (or subdirectory) we need to print the following information to the screen: name of the file (or subdirectory), inode number, file type, number of blocks, size in bytes, userid.

3) Write a **program p3** that will do **random access** (read operation) to a file where each access is of K bytes. We will run the program as: p3 K F. F is the name of the binary file which you will access in the program. Random access can be done by use of **lseek** function. You will use again Linux low-level I/O functions: open, **read**, write, close. In your program will measure the time it takes to do an access (a read operation of specified size). You can measure it using the **gettimeofday** function. Your program will do a lot of random accesses (say 100s). At the end, your program will print out the **average** random access time.

To do this, first create a large file. The run your program. Record the time output.

Reboot your machine and run the program again. Record the time output.

Clear the disk case (which is caching file data). You can do this by executing the following command:

```
sudo echo 3 > /proc/sys/vm/drop_caches
```

Then run your program again. Record the time output. Compare and interpret the results.

Include all outputs and your explanations, discussions into your report.

Submission

Put all your files into a project directory named with your ID (one of the IDs of team members for team projects), tar the directory (using **tar xvf**), zip it (using **gzip**) and upload it to Moodle. For example, a student with ID 20140013 will create a directory named 20140013, will put the files there, tar and gzip the directory and upload the file. The uploaded file will be 20140013.tar.gz. Include a **README.txt** file as part of your upload (including the name and ID of the student, at least – for team projects, names and IDs of all members will be included). Include also a **Makefile** to compile your programs. We want to type just **make** and obtain the executable and object files.

Additional Information and Clarifications

- This project does not require much coding. It requires quite a bit reading, self-learning, trying, and documenting.