COMP 3430 – Operating Systems – Lab 3
File systems

Purpose

The goal of this lab is to help you see how file systems concepts taught in class are manifest in a real operating system.

Introduction

Your lab will again take place in E2–420 EITC (where you will physically sit to do the lab) and will again use the Mac machines via remote login using ssh.

How to Prepare for the Lab

Your first step in preparation for the lab should be to read over the lab problems described below and make sure you understand them generally. The lab demonstrator will be available to answer specific questions during your actual lab. Your second step should be to review any relevant background material (e.g. the class notes, handouts, and web links specified below) as well as to read any man pages specified in this document.

Additionally, you might want to look at the code in the files `checkFile.c` and `mkLarge.c` from the *Examples* section of the course homepage in advance and make sure you understand them. Your final step in preparation for the lab should be to quickly review the relevant notes covered in class up to and including the material on file systems, to remember to bring the notes with you to your lab (since they may be useful), and to print a copy of this lab document and the hand–in document (if you have not already done so) to bring with you to the lab.

What to do?

Answer the questions below on the hand–in sheet. Be sure to hand in your completed sheet to the demonstrator before you leave the lab. This is how you will be graded!

Questions

[2] 1a. In linux, it is very easy to get information about files and disks. The `stat` command (which uses the stat system call) is very useful for this purpose. You can get the file system information for any file (including directories) with this command by `stat -f <file>`. Stat both the root directory of the system `/` and your home folder (hint: linux provides the home folder shortcut `~`, e.g., `cd ~`. What are the two different types? Use google to briefly explain what they mean.

[1] 1b. What are the block sizes of these devices (this is the smallest possibly writing/reading unit)? Why do you think would they be different?

2. You can also use various features of the `stat` system call directly in C. Download the `checkFile.c`, inspect it, and compile it. It is currently set to print a file's inode to screen. You can use this tool to investigate how file links work in linux.

   create an empty file called tmp1 by typing `touch tmp1`

   use the `ln` command to make both a hard link to this file (call it `tmp1-hardlink`) and a symbolic link (call it `tmp1-slink`) – check `man ln` for usage.

[2] 2a. use the checkFile program to investigate the inode #s of all three files. What did you notice? What does this tell you about hard and symbolic links?

[3] 2b. Next, modify `checkFile.c` to also print the number of hard links to the inode (check `man lstat`). Run the new program on all three files above as well as `checkFile.c` What did you find? Explain this result.

3. You will want to do the next steps on the primary file system of the linux machine. Create a folder in the temp directory (e.g., make /tmp/3430YOURID) and copy all of your files there.

Before copying make sure to set the permissions of the folder so that no one else can read it!! (`chmod -R 700 yourfolder`). Work in that directory for now.

Investigate how files are stored. You can create files of an arbitrary size using the `dd` command (no one knows what `dd` stands for, but there are some theories). `dd` simply performs raw reads and writes to files: `dd if=<in file> of=<out file> bs=<block size> count=<# of blocks>`. Linux has a special `/dev/zero` file that will provide infinite zeros, so you can use `dd` with `if=/dev/zero` to create files full of 0s. The `bs` that you use is arbitrary and simply tells `dd` what size to use in reads and writes. Changing the `bs` does not impact your final file, and your final size is simply `count*bs`.

Use `dd` to make a file 512 bytes (called `512`), 2048 (called `2k`), 4096 (called `4k`), 4097 bytes (called `4k-plus`). use `ls -al` to confirm the sizes are correct.

Following, update `checkFile.c` to also print the number of blocks that a file is using on disk. (again check the man page for `lstat`).

[3] 3a. Check the number of blocks used by each file, as well as the 0-size tmp1 file created earlier. What did you find? Can you explain?

[1] 3b. How can this data explain the block size of the file system?

[2] 3c. Fundamentally (discounting the file system), how large should a file be before it requires 24 fundamental 512b blocks? Given your above answers, on this system how large will a file actually be before it requires 24 blocks? Why is this? make sure to use `dd` and `checkFile` to test your theory.

[1] 3d. Does this waste or save space on the HD?

[2] 3e. now, copy all your files back to a directory in your home directory and perform the same checks. What is different. Why? How does this agree / disagree with findings from question 1b?

[2] 4a. Work again on the tmp folder on the machine's disk. Download the program `mklarge.c` from the examples page and compile it. Open the source and carefully inspect what is happening. If you run this program, it will create a file that is quite large. Run `checkFile.c` on this and report the number of blocks that this file is taking. What did you find? Why is this the case?

[4] 5a. What is in a link? Write a program from scratch called `checkLink.c` that takes an argument and uses the `readlink` system call (check its man page) to read the raw contents of a symbolic link. Make sure to do basic error checking: incorrect number of arguments, and if `readlink` returns an error, use `perror` to print out what the error was. If there is no error, print the contents of the link to screen as a string. Hint: `readlink` does not null terminate!

Use your program on `tmp1-link` and `tmp1-hardlink`. What did you find? Why don't links just store an inode?

Demonstrate your link and checkFile programs for the lab demonstrator. The demonstrator will initial and make any comments on the hand–in sheet. If you are not able to demonstrate a working program during lab time then notify your TA and when finished submit your completed program using the dropbox on the D2L by the end of the day – if you do demonstrate, then a dropbox submission is not necessary.