Carleton University
School of Computer Science

COMP 3000 (FALL 2020) OPERATING SYSTEMS ASSIGNMENT 2

Please submit the answers to the following questions via CULearn by 23:59 on November 3, 2020. There are 18 points (weight: 0.25) in 8 questions in total.

Submit your answers as a **gzipped tarball** "*username*-comp3000-assign2.tar.gz" (where username is your MyCarletonOne username). Unlike tutorials, assignments are graded for the correctness of the answers.

The tarball you submit **must** contain the following:

1. A **plaintext** file containing your solutions to all questions, including explanations. Further details are provided below.
2. A README.txt file listing the contents of your submission as well as any commands the TAs should use to compile your code (or provide a Makefile). The TAs will only try with available information (e.g., the default is gcc with no special options).
3. The source code for your modified version of 3000test. This file should include all required changes for one question. You should submit one per applicable question (i.e., one file for P2Q2 and one file for P2Q5). **This way, if one has errors it will not affect the other question's marks.**
4. A diff file showing the modifications, by comparing each submitted file in 3 above and the original: `diff -c 3000test.c your_modified_3000test2.c > 3000test2.diff`; `diff -c 3000test.c your_modified_3000test5.c > 3000test5.diff`. Avoid moving around or changing existing code (unless necessary) which may be distracting.

You can use this command to create the tarball: `tar zcvf username-comp3000-assign2.tar.gz your_assignment_directory`. ****Don't forget to include your plaintext file!!****

No other formats will be accepted. Submitting in another format will likely result in your assignment not being graded and you receiving no marks for this assignment. In particular, do not submit an MS Word, OpenOffice, or PDF file as your answers document!

Empty or corrupted tarballs may be given a grade of zero, so please **double check your submission** by downloading and extracting it.

Don't forget to include what outside resources you used to complete each of your answers, including other students, man pages, and web resources. You do not need to list help from the instructor, TA, or information found in the textbook.

Use of any outside resources verbatim as your answer (like copy-paste or quotation) is not allowed, and will be treated as unauthorized collaboration (if it's from another student).

Please do **NOT** post assignment solutions on Discord or cuLearn forums or it will be penalized.

Questions – part 1 [6]

1. [1] Can you use the inode number to uniquely identify a file? Why or why not?
2. [2] In an empty directory, if you do the following:
   `touch file1; ln file1 file2; ln -s file2 file3; mv file1 file4`
   Are you able to edit file3? Why or why not?
   Now, you `mv file2 file5`. Are you able to edit file3? Why or why not?
   Be more concrete than just "because filex is a yyyy".

3. [3] Mention two obvious reasons why the password of a user on the course VM cannot be easily retrieved by unauthorized parties. Assume that the unauthorized party already has access to your VM, as another non-root user.

Questions – part 2 [12]

The following questions will be based on the **original** 3000test.c in Tutorial 5:

> *A hole in a file is an area represented by zeros. It is created when data is written at an offset beyond the current file size, or the file size is "truncated" to something larger than the current file size. The space between the old file size and the offset (or new file size) is filled by zeros. Most filesystems are smart enough to mark the holes in the inode, and not store them physically on disk (these are also known as sparse files).*

Create a sparse file (file with "holes") with: `truncate -s 256M myfile`
Create another one: `dd if=/dev/zero of=myfile2 bs=1M count=1 seek=255`
Create a third one: `dd if=/dev/null of=myfile3 bs=1M count=1 seek=256`
Examine the size of the files with `ls -lais`

1. [3] Explain why the logical size and the physical size are what they are now, for myfile, myfile2 and myfile3 respectively. Hint: make proper use of `man` pages for the commands/devices involved.
2. [3] Add a function to 3000test.c so that it also prints a line to indicate whether the file has "holes", e.g., "This file has/does not have holes". You can name the function as you like (e.g., `sparsecheck()`) but it must be a single function called in `main()`.
   It should **only** be called if the file is a regular file or a symbolic link. For symbolic links, resolve them to the actual files pointed to.
   Hint: you can compare the two sizes to achieve the purpose.
   Make sure your code works by testing it with at least a few typical files.
3. [2] Explain why if logical size is greater than physical size there must be holes, or put another way, this file must be a sparse file. Keep in mind that space is allocated at the granularity of blocks (e.g., 4K)
4. [1] What does 3000test.c call `mmap()` for (the purpose of line 59)?
5. [3] The lines 54 – 79 are to count the occurrences of the character 'a' if the file is a regular file. Change the code to avoid using `mmap()` (line 59) but not affect the function. So, the program should still output the right number of 'a's in a regular file. Hint: consider `read(2)` or `pread(2)`.