**README File**

Attachments: project\_directory\_structure.txt 🡪 provides the directory tree for the project folder

Note: In the imageFiles folder, fat16FS\_full200MB.dmg is the only mountable filesystem. All the other images are portions of this entire file system image I created for small test cases.

fat16FS\_full200MB.dmg can be mounted, files saved to it, unmounted and then processed using the instructions below.

There are 4 C-source files and should be used in the order below:

1. strippingNullBytes.c 🡪 this file takes a file system as a .dmg file (from imageFiles folder), strips all the null bytes and returns a “null-free” .dmg file. It also returns a mapFile (.txt) with the start address and length of every null-byte sequence that was stripped from the file.
2. hidingNoNullImg.c 🡪 The program takes as input the mapFile (.txt) the null-free .dmg file generated by strippingNullBytes.c along with a coverfile list(.txt file) that has the full-path of files that can be used as cover files along with the available slack space for each potential coverfile.
3. retrievingNoNullImg.c 🡪 The program takes as input the coverfile list(.txt file) used by 'hidingNoNullImg.c' with the available slack space for each coverfile used. It reassembles the contents of all the cover files OR up to a specified number of bytes and returns a .dmg file saved to dstFullPATH.
4. restoringNullBytes.c 🡪 The program takes as input a mapFile created by 'strippingNullBytes.c' and the .dmg file returned by 'retrievingNoNullImg.c'. It then replaces all the null bytes that were stripped (which is noted in the mapFile) and returns a dmg file that is (should be) an exact replica of the original dmg file used by 'strippingNullBytes.c'

There are 4 bash scripts (just one line shell code), which are exactly the same but each is meant to enumerate the files and their slack spaces for a specific top-level folder.

1. calc\_slack\_usr.bash 🡪 recursively enumerates all files under /usr, computes the slack space of each file and returns a txt file names “usrTotalslack.txt”with the file name (with full path) and its available slack.
2. calc\_slack\_lib.bash, calc\_slack\_sys.bash, calc\_slack\_dev.bash, calc\_slack\_etc.bash, calc\_slack\_boot.bash 🡪 each one is same as the above except thay each focus of a separate root level folder.
3. IMPORTANT NOTE: the output file has the file name (with full path) and available slack space on separate lines. There is a separate python script to process this usrTotalslack.txt file. Note that the file has certain entries that need to be excluded when processing with python script.

There is one python script:

1. processCoverFileList.py 🡪 processes the output from the above bash script(s). It will create a new file by combining the file name (with full path) and its available slack (which are on separate lines when output by bash) into single line tab separated entries. During the processing, it will also eliminate any file that is not a regular file. Additionally, in its current form, it eliminates files that have a slacks space <2000bytes, >4095 bytes. This helps avoid using slackspaces that are very small. The output file can also be sorted (by opening it in excel) to sort coverfiles from largest to smallest slack spaces.

Steps you need to follows:

1. run the bashscript “calc\_slack\_usr.bash” on your local Ubuntu instance to generate “usrTotalSlack.txt” output file
2. run the “processCoverFileList.oy” to process the output from step-1. This will generate 3 separate files – usrCoverFileSize.txt, usrCoverFileName.txt and usrCoverFileList.txt. You will use usrCoverFileList.txt for the next 4 steps where needed
3. compile and run strippingNullBytes.c. this will output a nonNull appended to the input dmg file you provide in the imageFiles folder and also a mapfile under the mapFiles folder
4. compile and run hidingNoNullImg.c and this will hide the noNull dmg from step-3 using the usrCoverFileList.txt from step-2
5. compile and run retrievingNoNullImg.c and this will generate a new image in the imageFiles folder with “retrieved” appended to the file name. It will use the usrCoverFileList.txt for reference. This is the inverse of step-4.
6. Compile and run restoringNullBytes.c and this will generate a new image in the imageFiles folder with “restored” appended to the file name. It will use the mapFile generated in step-3 to restore the null bytes that were stripped. This is the inverse of step-3.
7. If you compute the hash of the original filew system dmg file used in step-3 and the final filestystem dmg file generated in step-6 (with restored appended to file name), the two should match. Also the dmg fine generated at the end of step-6 should be mountable.