Description

dos_scandisk.c is a scandisk or a filesystem checker for a DOS (FAT12) filesystem on a floppy disk. When compiled, it is run by typing "./dos_scandisk <image_name>" to the command line, for example "./dos_scandisk badfloppy2.img".

The scandisk program first memory maps the image file using mmap_file, and stores the memory map in the variable image_buf. It then scans the boot sector and gets metadata from its bpb, which is stored in the bpb variable, which is of type struct bpb33. Both the image_buf and bpb variables will be used for other functions later in the program.

It then recursively traverses all directories contained in the image, starting from the root directory (which always starts at cluster 0), which is done with the follow_dir function in dos_scandisk.c. It also 'visits' each cluster that makes up the directory file (using an array that stores information for each cluster, whether it has been visited or not). For each non-directory file it detects, it 'visits' the clusters that make up the file using the follow_non_dir function. Afterwards, information about the file name and extension, its size in bytes, starting cluster, and number of clusters that make up the file is stored in an array (files) of struct file.

Afterwards, it iterates through every cluster in the image. If a cluster contains part of a file that we have not visited beforehand, we mark it as the starting cluster of an unreferenced file. It then 'visits' the clusters that make up the unreferenced file and print them out using the follow_unreferenced function. Also, information about the file name and extension, size in bytes, starting cluster, and number of clusters that make up the file is stored in a separate array (unref) of struct file.

For each unreferenced file, the name, start cluster, and number of clusters in the file is printed. Then, a new directory entry is created that links to the unreferenced file, which is appended to the end of the root directory using the append_de function.

Then, for each file, it checks if the size (in bytes) in the file's directory entry is consistent with number of clusters it uses in the FAT. If they are inconsistent, it prints information about the file (name, extension, size of the file from its directory entry, total size of all clusters used by the file). It then frees clusters beyond the end of file as defined in its directory entry, which is done by going to the cluster where the end of file should be, setting its FAT entry to 4095, and setting the FAT entries of all clusters in the file after it to 0. This is done by the change_last_cluster function.