Λειτουργικά Συστήματα(Κ-22)
Τμήμα Περιττών
Ιανουάριος 2021
Τροιζή Θεοδώρα
sdi1800197

quic.c : includes the main. We keep the time at the beginning and at the end of the program, and we compute the running time of the program.
main calls copy_directory and prints the statistics.

files.c: includes all the functions that handles the files.

copy_files: A function that reads from a file and copies its content to another file.

create_file: creates a file

*s_file:*checks the mode of the file. If it is a regular file returns 1.
compare_files: Compares 2 files based on the criteria of the project(size of file and last modified time).

directories.c: includes all the functions that handles directories.

*copy_directory:*returns a struct that contains the statistics of the program.
We create 4 bi-directional lists , where we keep the files/directories names. The consept is that we compare every file/directory from the source to every file/directory from the destination. We don't want to opendir and closedir everytime, so we loop through each directory only once and store only the name of the files/directories to lists.We don't store the whole path, because later on we want to compare *only* the names of the files(strcmp). The paths are easy to be made , because we know the source and destination directory.
If the destination directory doesn't exist, we create it.
We iterate through the source_files list and we check if the same name exists on the destination_files list. If the name exists, we compare the two files. If the criteria are satisfied we continue to the next file , else we copy the source to the destination file. We remove the file's name from the destination's files list. Our goal is that at the very end, the only files left in the destination list are those that we have to delete, because they don't exist in source. If the name doesn't exist in the destination list, we create a new file then copy the data.
We iterate through the source_directories list and we check if the same name exists on the destination_directories list. If the name exists, we compare the two directories recursively. If the criteria are satisfied we continue to the next directory , else we copy the source to the destination directory. If the directories were the same ,we remove the directory's name from the destination's directories list. Our goal is that at the very end, the only directories left in the destination list are those that we have to delete, because they don't exist in source.

If the flag "delete" is equal to 1, we must delete all the files/directories from the destination that doesn't exist in source. The deletion of directories is completed recursively. We empty all of its content then delete the directory.
At the end we free the allocated memory.

*create_directory:*creates a directory.

*remove_directory:*a recursive function to delete a directory. We iterate through directory and if we find a file we delete it, if we find a directory we call the same function, until nothing is left.

is_directory: checks the mode of the file. If it is a directory returns 1.

List.c: It is a simple implementation of ta bi-directional list with 2 key points: i) we use a "dummy" node in order to make some algorithms more simple(we don’t have to make an extra case if the list is empty, because it will never be completely empty.) ii) The value of list-node is of type void*. That is because we use the same implementation of the List for many types of data(int, char etc) and we want a general type to cover them all.
The struct blist contains a dummy node, the last node and its size. The blist_node contains 2 pointers: one on its previous node and one on its next node.

blist_insert(): If the list is empty, the last node points to dummy, so when the new node is added the last node must points to new node. Apart from that, it's a classical implementation of a list, except for the fact that when a new node is added we must link it both with its previous and its next. After the insertion, we update the size adn the last node.

blist_remove() We want to remove the node. So, we must link its previous node with its next. Special cases are if the node is the first or last in the list. On the first case, we must link it with the dummy node, on the second case, the last node is removed, and its previous must now point to NULL.