Λειτουργικά Συστήματα – Τμήμα Περιττών 2018/2019
Εργασία 4

ΜΑΡΑΓΚΟΖΑΚΗΣ ΓΕΩΡΓΙΟΣ – 1115201500089

Δομή αρχείων εργασίας:

mirr.c functions.c functions.h structures.c structures.h Makefile

Δομές Δεδομένων:

Δέντρα ονομάτων: **typedef struct TreeName**
Λίστα δεικτών σε παιδία κόμβου Δέντρου: **typedef struct TreeNameList**
Λίστα Inode αρχειων: **typedef struct InodeList**
Λίστα ονομάτων αρχείων σε Inode: **typedef struct NameList**
Λίστα καταλόγων που τελούν υπό παρακολούθηση: **typedef struct wd_array**

Αρχεία συναρτήσεων:

structures.c :

void add_TreeNameList_node(TreeNameList*, TreeName*) : Παίρνει σαν όρισμα τη λίστα του συγκεκριμένου καταλόγου και του προσθέτει στο τέλος της λίστας ένα νέο κόμβο TreeName.

TreeNameList *find_TreeNameList_node(TreeNameList*, TreeName*) : Διατρέχει αναδρομικά τη δοσμένη λίστα και βρίσκει το συγκεκριμένο TreeName, και επιστρέφει ένα δείκτη στη θέση της λίστας που βρέθηκε.

TreeNameList *remove_TreeNameList_node(TreeNameList*, TreeName*) : Διατρέχει τη δοσμένη λίστα και αφερεί από αυτή τον κόμβο TreeName, επιπλέον επιστρέφει την νέα κεφαλή της λίστας αν διαγράφηκε το πρώτο στοιχείο.

add_TreeName_node(TreeName*, char*) : Για το δοσμένο TreeName του προσθέτεται το name του.

void add_child_TreeName_node(TreeName*, TreeNameList*, char*) : Δημιουργείται ένας νέος κόμβος TreeName, αρχικοποιείται με το name και καλείται η add_TreeNameList για να προστεθεί στη δοσμένη λίστα.

TreeNameList *remove_child_TreeName_node(TreeName*, TreeNameList*, char*) : Δημιουργείται και αρχικοποιείται ένας temporary κόμβος TreeName και καλείται η revove_TreeNameList_node για να βρεθεί και να αφαιρεθεί ο αντίστοιχος κόμβος της Λίστας. Στο τέλος αποδεσμεύεται κι αυτός.

TreeNameList *find_child_TreeName_node(TreeName*, TreeNameList*, char*) : Δημιουργείται και αρχικοποιείται ένας temporary κόμβος TreeName και καλείται η find_TreeNameList_node για να βρεθεί ο αντίστοιχος κόμβος της Λίστας. Στο τέλος αποδεσμεύεται.

void remove_TreeName_node(TreeName*, char*) : Διαγράφεται ο κόμβος TreeName, και καλούνται οι αντίστοιχες συναρτήσεις για την διαγραφή των περιεχόμενων του όπως, η λίστα παιδιών του, αν είναι κατάλογος και το inode του αν είναι το μοναδικό όνομα που δείχνει προς αυτό.

void add_InodeList_node(InodeList *, struct stat*) : Όπως και στις άλλες λίστες, αναδρομικά προσθέτονται τα στοιχεία στη λίστα.

void Init_InodeList_name(InodeList *, char*) : Αρχικοποιούνται το όνομα και αυξάνεται ο δείκτης ονομάτων που δείχνουν εδώ.

void Init_InodeList_node(InodeList *, char*) : Αρχικοποιούνται τα στοιχεία του Inode που πάρθηκαν για το αρχείο μέσω της stat.

void remove_InodeList_name(InodeList *, char*) : Αφαιρείται από τη δομή Inode ο κόμβος με το name, (το name κρατείται ως full path για να μην υπάρχουν διπλότυπα).

Αντίστοιχα οι συναρτήσεις find, delete, κτλπ είναι ίδιας λογικής και για την NameList

functions.c :

ΔΗΜΙΟΥΡΓΙΑ ΚΑΙ ΒΗΜΑ 1

TreeName *create_source(TreeName*, char*) : Καλούνται οι αντίστοιχες συναρτήσεις από το structures.c και δημιουργείται ο αρχικός TreeName κόμβος. Στη συνέχεια για αυτόν τον κόμβο καλείται η create_children η οποία διαβάζει και δημιουργεί κόμβος για τα περιεχόμενα του source καταλόγου.

void create_children(TreeName*, char*, InodeList*) : Με αναδρομικό τρόπο διαβάζονται όλα τα περιεχόμενα του αρχικού καταλόγου και δημιοργείται το δέντρο ονομάτων και η αντίστοιχη λίστα των inode. Στο τέλος για κάθε λίστα παιδιών καταλόγων καλείται η quicksort, στην οποία ταξινομούνται οι κόμβοι αδερφοί.

void quicksort_TreeNameList(TreeNameList *start, TreeNameList *finsh) : Τυπική υλοποίηση quicksort για την ταξινόμηση των κόμβων αδελφων της κάθε λίστας TreeNameList. Η υλοποίηση της βασίστηκε σε προηγούμενη υλοποίηση αυτής για εργασία στο μάθηκα ΥΛΟΠΟΙΗΣΗ ΣΥΣΤΗΜΑΤΩΝ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ.

void go_depth(TreeName *src, TreeName *bkp, InodeList *i_src, InodeList *i_bkp) : Διατρέχονται τα δύο δέντρα ονομάτων κατά βάθος και σε κάθε κόμβο κατάλογο υλοποιείται το τμήμα α του αλγορίθμου συγχρονισμού και στη συνέχεια το τμήμα β. Αναλυτικότερα, για κάθε κατάλογο που επισκεπτεται δημιουργούνται στο backup όσοι κατάλογοι υπάρχουν και στο source και αντίστοιχα διαγράφονται όσοι δεν υπάρχουν στο source. Στη συνέχεια αφού ικανοποιηθούν τα παραπάνω, για κάθε κόμβο κατάλογο στο βάθος που βρίσκεται η συνάρτηση υλοποιούνται τα τμήματα γ, δ, και ε, για να δημιουργηθούν, διαγραφούν και τελικά προστεθούν στη backup δομή τα αρχεία που βρίσκονται και στο source. Η συνάρτηση για να φτάσει στο βάθος του δέντρου χρησιμοποιεί αναδρομική υλοποίσηση.

TreeNameList *sync_algorithm_step_a(TreeName *src, TreeName *bkp, InodeList *i_src, InodeList *i_bkp) : Για το συγκεκριμένο βάθος καταλόγου ελέγχονται ποιοι κατάλογοι του source δεν υπάρχουν στο backup και αντίστοιχα δημιουργούνται στον κατάλογο backup και προσθέτονται στη δομή του. Επιπλέον, ελέγχεται και αφαιρείται από το backup, οποιδήποτε αρχείο έχει όνομα ίδιο με καταλόγου στο source.

TreeNameList *sync_algorithm_step_b(TreeName *src, TreeName *bkp, InodeList *i_src, InodeList *i_bkp) : Για το συγκεκριμένο βάθος καταλόγου ελέγχονται ποιοι κατάλογοι του backup δεν υπάρχουν στο source και αντίστοιχα καταργούνται από τον κατάλογο backup και αφαιρούνται από τη δομή του. Επιπλέον για κάθε κατάλογο που διαγράφεται γίνεται έλεγχος για τα περιεχόμενα του και διαγράφονται κι αυτά μαζί με τον κατάλογο. Οι λειτουργίες αυτές επιτελούνται μέσω της

συναρτήσεως, delete_directory_contains, η οποία λειτουργώντας ανδρομικά διαγράφει φυσικά και από τη δομή backup τα περιεχόμενα του υπό διαγραφή καταλόγου.

TreeNameList *sync_algorithm_step_c(TreeName *src, TreeName *bkp, char *name, InodeList *i_bkp) : Για το υπάρχων βάθος καταλόγου, και αφού προηγούμενως στην go_depth ελέγχθηκε για ένα όνομα αρχείου του source ότι δεν υπάρχει στο backup, υλοποιούνται οι απαιτήσεις του συγκεκριμένου βήματος και ελέγχεται αν υπάρχει αντίγραφο του source στο backup και συνδέεται με το δομή inode, αλλίως δημιουργείται νέο αρχείο και τοποθετείται στη δομή. Η αντιγραφή του περιεχομένου του αρχείου από το source στο backup υλοποιείται μέσω της copy_source_to_backup, η οποία βασίστηκε σε ένα από τα παραδείγματα του Κύριου Δελή.

TreeNameList *sync_algorithm_step_d(TreeName *src, TreeName *bkp) : Αφού, εντοπιστεί, μέσω της look_for_filename_in_source, ότι το συγκεκριμένο αρχείο δεν υπάρχει στο source αφαιρείται από το backup και απο τη δομή του.

TreeNameList *sync_algorithm_step_e(TreeName *src, TreeName *bkp) : Αφού ελεγχθεί από την go_depth ότι τα metadata είναι ίδια τότε συνδέεται το inode του source με αυτό του προορισμού.

ΒΗΜΑ2

void *start_inotify(TreeName *source, TreeName *backup) : Καλείται η add_to_inotify, μέσω της οποίας προσθέτονται αναδρομικά όλοι οι κατάλογοι του source στη δομή wd_array και θέτονται υπό παρακολούθηση. Στη συνέχεια, επιλέγοντας από τις διαθέσιμες επιλογές το m αρχίζει η παρακολούθηση, καλείται η monitoring, των γεγονότων που μπορούν να συμβούν εντός των καταλόγων του source. Οι υπόλοιπες επιλογές έχουν βοηθητικό χαρακτήρα, και μπορούν να αγνοηθούν.

wd_array *monitoring(TreeName *source, int fd, wd_array *wd_arr, TreeName *backup, Inode *i_src, char *name) : Η συνάρτηση αυτή εκτελεί χρέη “ΜΕΓΑΛΟΥ ΑΔΕΛΦΟΥ” καθώς για κάθε γεγονός που συμβαίνει εντός των καταλόγων, ανάλογα το γεγονός επιτελεί την αντίστοιχη λειτουργία. Η λειτουργία self_delete έχει υλοποιηθεί εντός αυτής της συνάρστησης.

TreeName *in_create(TreeName *source, int fd, wd_array *wd_arr, TreeName *backup, Inode *i_src, char *name) : Στην περίπτωση δημιουργίας ενός νέου αρχείου σε κάποιο κατάλογο εντός του source τότε μέσω της get_TreeName_node εντοπίζεται ο κατάλογος μέσα στον οποίο συντελέστηκε η δημιουργία ενός αρχείο ή καταλόγου και αντίστοιχα με τον τύπο του δημιουργηθέντως καλείται το τμήμα α του αλγορίθμου συγχρονισμού για κατάλογο, ή αντίστοιχα το τμήμα c για αρχείο.

TreeName *in_delete(TreeName *source, int fd, wd_array *wd_arr, TreeName *backup, Inode *i_src, char *name) : Αφού βρεθεί μέσω της monitoring αν πρόκειται για αρχείο ή κατάλογο τότε στην περίπτωση αρχείου, μέσω αυτής της συνάρτησης καταργείται και διαγράφεται το αρχείο από το backup. Αν πρόκειται για κατάλογο τότε υλοποιείται η ίδια λειτουργία με τη διαφορά ότι τώρα ο κατάλογος πρέπει να αφαιρεθεί και από τη δομή παρακολούθησης καταλόγων. Αυτό επιτελείται μόνο αν ο κατάλογος είναι άδειος εσωτερικά.

TreeName *get_TreeName_node(TreeName *t, char *name) : Διατρέχεται από την αρχή όλο το δέντρο όνομάτων του source, κατά βάθος, και εντοπίζεται ο κατάλογος με το name, το οποίο αποτελεί το full pathname.

Σύντομη περιγραφή υλοποιημένων και μη λειτουργιών :
Έχουν υλοποιηθεί, το Βήμα 1 και μερικώς το Βήμα 2.

Όσον αφορά το Βήμα 1, ικανοποιούνται όλες οι απαιτήσεις της εκφώνησης, ενώ χρησιμοποιούνται τα αντίστοιχα system calls για τη δημιοργία, διαγραφή κτλπ των αρχείων και καταλόγων. Ενδεικτικά, μόνο στη λειτουργία mkdir χρησιμοποιήθκε η αντίστοιχη εντολή μέσω μιας νέας διεργασίας με τη χρήση fork και execlp.

Όσον αφορά το Βήμα 2, έχουν υλοποιηθεί επιτυχώς οι λειτουργίες CREATE, DELETE και SELF_DELETE. Ενώ δεν έχουν υλοποιηθεί τα IN_ATTRIB, IN_MODIFY, IN_CLOSE_WRITE, IN_MOVED_FROM, IN_MOVED_TO.

Υλοπόιση Εργασίας:

Η υλοποίση της εργασίας έγινε ατομικά από εμένα. Στην παραδοτέα εργασία συμπεριλαμβάνονται ορισμένα κομμάτια κώδικα από τα παραδείγματα του Κύριου Δελή, όσον αφορά την inotify, και το διάβασμα καταλόγων. Επιπλέον στα παραδοτέα συμπεριλαμβάνονται 2 κατάλογοι για την εκτέλεση της εργασίας.

Εκτέλεση:

Κατά την εκτέλεση της εργασίας δεν διαπιστώθηκαν errors ή memory leaks μέσω της valgrind. Επιπλέον, χρησιμοποιείται Makefile για την μεταγλώττιση και εκτέλεση της εργασίας.

Ενδεικτική εντολή εκτέλεσης:

./mirr mydir mybackup