PROJECT 4

Λιλίτσης Ιορδάνης 2867 liordanis@uth.gr
Σαπουντζή Αθανασία Δέσποινα 2624 asapountzi@uth.gr
Ξωχέλλη Ελένη 2761 xeleni@uth.gr

Αρχιτεκτονική του συστήματος

Στην εργασία φτιάξαμε ένα file system αξιοποιώντας το FUSE filesystem. Σκοπός ήταν η «συμπίεση» του χώρου αποθήκευσης μέσω του εντοπισμού και επαναχρησιμοποίησης κοινών blocks.

Όσον αφορά την αρχιτεκτονική του συστήματος διατηρούμε στο δίσκο κάθε νέο block σε ξεχωριστό file στο directory **.realStorage** που αποτελεί την «αποθήκη» μας. Για να έχουμε εύκολη πρόσβαση κρατάμε σε μία δομή πληροφορίες για κάθε block και για κάθε file.

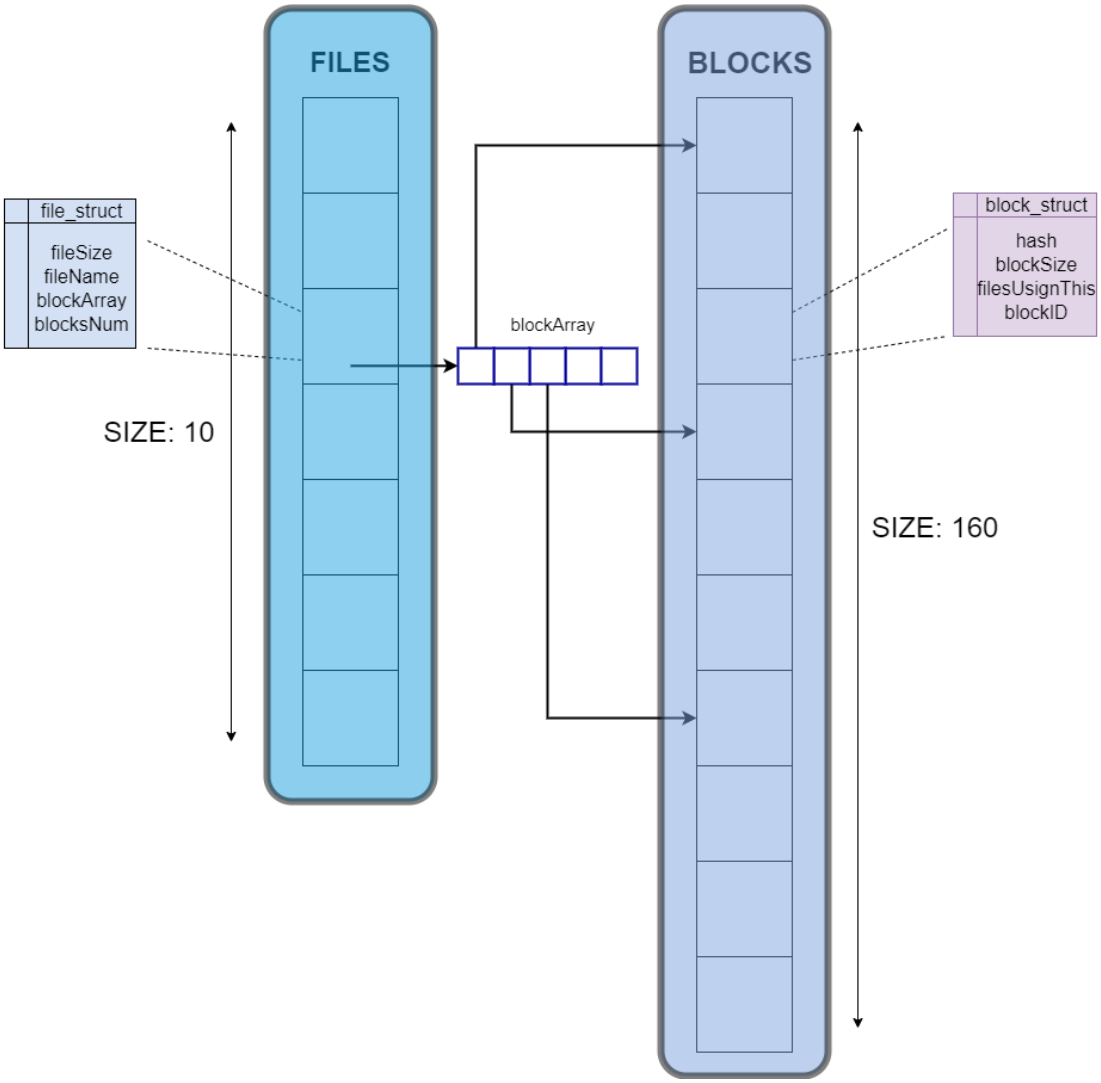
Πιο συγκεκριμένα **για κάθε block** κρατάμε σε struct:

- **unsigned char hash[]**: μοναδικό για το block και βοηθάει στο διαχωρισμό τους
- **size_t blockSize**: μέγεθος του block
- **int filesUsingThis**: αριθμός των files που χρησιμοποιούν το block
- **int blockID**: μοναδικό ID που αντιστοιχίζει το hash με το file

Επίσης **για κάθε file** κρατάμε σε struct:

- **char fileName[PATH_MAX]**: το path του file
- **struct block_struct* blockArray[16]**: πίνακας από pointers σε block structs του συγκεκριμένου file
- **int fileSize**: μέγεθος του file
- **int blockNum**: αριθμός των blocks του file

Στο παρακάτω σχήμα βλέπουμε τους δυο πίνακες για **file structs** και **block structs** με τα μεγέθη τους και τα περιεχόμενα του struct. Οι πίνακες αυτοί συνδέονται μέσω των blockArray των files που παρέχει pointers στα blocks.



Συναρτήσεις που τροποποιήθηκαν

Για την υλοποίηση που περιγράφηκε προσθέσαμε ορισμένες αρχικοποιήσεις στην main και τροποποιήσαμε κατάλληλα τις παρακάτω συναρτήσεις του BBFS:

- **int bb_write(const char *path, const char *buf, size_t size, off_t offset, struct fuse_file_info *fi):** ελέγχουμε αν υπάρχουν ήδη το block και το file. Σε περίπτωση που δεν υπάρχει το file δημιουργούμε το file struct στην πρώτη κενή θέση της δομής. Αντίστοιχα δημιουργούμε και το block struct αν δεν έχει βρεθεί block με ίδιο hash και ανοίγουμε νέο file για την αποθήκευση των δεδομένων του. Αφού πλέον υπάρχουν (είτε τα δημιουργήσαμε είτε προϋπήρχαν) ανανεώνουμε κατάλληλα την παραπάνω δομή. Επιστρέφουμε τον αριθμό των bytes που γράψαμε.

- **int bb_read(const char *path, char *buf, size_t size, off_t offset, struct fuse_file_info *fi):**

ελέγχουμε αν υπάρχει το file και αναζητούμε το block στην κατάλληλη θέση στο blockArray. Αφού βρούμε το block struct, ανοίγουμε το file και διαβάζουμε τα δεδομένα. Επιστρέφουμε τον αριθμό των bytes που διαβάσαμε. Σε περίπτωση που επιθυμούμε να διαβάσουμε περισσότερα bytes από το BLOCK_SIZE (4KB), διαβάζουμε επαναληπτικά όσα blocks χρειάζονται για να φτάσουμε το επιθυμητό size (bbfs.c:408).

- **int bb_unlink(const char *path):** ελέγχουμε αν υπάρχει το file και όταν το βρούμε μειώνουμε το filesUsingThis των blocks που χρησιμοποιούνται από το file και διαγράφουμε όσα blocks έχουν μηδενική τιμή του filesUsingThis. Σημειώνουμε το κελί του πίνακα για τα files ως κενό.

- **void bb_destroy(void *userdata):** Απελευθερώνουμε τη μνήμη που δεσμεύσαμε για τις πληροφορίες για τα blocks και τα files.

- **int bb_getattr(const char *path, struct stat *statbuf):** Αναζητούμε το πραγματικό μέγεθος του file και το αποθηκεύουμε στο struct που χρησιμοποιεί η lstat.

Για τα παραπάνω υλοποιήσαμε και κάποιες βοηθητικές συναρτήσεις που βρίσκονται στο myfslib.c.

Πειραματική Αξιολόγηση

Για να δοκιμάσουμε το σύστημά μας φτιάξαμε ένα απλό script για αντιγραφή αρχείων (test.sh). Πρώτα δημιουργούμε δυο αρχεία μεγέθους 8KB και στη συνέχεια αντιγράφουμε τα περιεχόμενα τους σε δύο νέα.

Αρχικά τα πρώτα δύο αρχεία περιέχουν το καθένα από 2 φορές το ίδιο block των 4KB. Έτσι δημιουργούνται και αποθηκεύονται στο .realStorage μόνο δυο νέα block των 4KB (0.txt και 1.txt) ενώ ο χώρος που φαίνεται να καταλαμβάνουν τα αρχεία είναι 16KB.

Μετά την αντιγραφή σε δύο νέα αρχεία στο .realStorage παραμένει ίδιο (8KB) ενώ ο χώρος που φαίνεται να καταλαμβάνουν είναι 8*4=32KB.

Παρακάτω βλέπουμε αναλυτικά τα αρχεία μετά την αντιγραφή καθώς και τα blocks που είναι αποθηκευμένα στο .realStorage.

```
user@oslab:~/project4/fuse-tutorial-2018-02-04/example$ ./test.sh
user@oslab:~/project4/fuse-tutorial-2018-02-04/example$ cd mountdir/
user@oslab:~/project4/fuse-tutorial-2018-02-04/example/mountdir$ ls -l
total 0
-rw-rw-r-- 1 user user 8192 Jun 12 17:27 test1.copy
-rw-rw-r-- 1 user user 8192 Jun 12 17:27 test1_extended.output
-rw-rw-r-- 1 user user 8192 Jun 12 17:27 test2.copy
-rw-rw-r-- 1 user user 8192 Jun 12 17:27 test2_extended.output
user@oslab:~/project4/fuse-tutorial-2018-02-04/example/mountdir$ cd ../.realStorage/
user@oslab:~/project4/fuse-tutorial-2018-02-04/example/.realStorage$ ls -l
total 8
-rw-rw-r-- 1 user user 4096 Jun 12 17:27 0.txt
-rw-rw-r-- 1 user user 4096 Jun 12 17:27 1.txt
```

Για την εκτέλεση αυτής της πειραματικής αξιολόγησης:

- Τρέχουμε το Makefile που βρίσκεται στο /filesystem/fuse-tutorial-2018-02-04/src.
    - Προσοχή, το Makefile στον φάκελο /filesystem/fuse-tutorial-2018-02-04/example επαναφέρει τους φακέλους στην αρχική τους κατάσταση. Έτσι λοιπόν δεν πρέπει να τρέξει
- Μεταφέρουμε τα δυο scripts (run.sh, test.sh) από το experiments μέσα στο φάκελο /filesystem/fuse-tutorial-2018-02-04/example
- Τρέχουμε το run.sh για να γίνει mount ο φάκελος mountdir και να τεθεί σε λειτουργία το file system μας.
- Τρέχουμε το test.sh για να δούμε τα παραπάνω αποτελέσματα.