Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών
Λειτουργικά Συστήματα (ΗΥ321)

Ακαδημαϊκό Έτος 2018-2019

4η Εργαστηριακή Άσκηση

Τελευταία Ενημέρωση: 20 Μαΐου 2019

- BBFS Makefile with libssl support

Περιεχόμενα

1 Εισαγωγικά

Στόχος αυτής της εργασίας είναι να αντιληφθείτε πώς υλοποιείται και δουλεύει ένα απλό σύστημα αρχείων στο Linux. Πιο συγκεκριμένα, θα κληθείτε να υλοποιήσετε ένα σύστημα αρχείων το οποίο προσπαθεί να ”συμπιέσει” τα δεδομένα του, αναγνωρίζοντας ευκαιρίες διαμοίρασης blocks μεταξύ διαφορετικών αρχείων (ή και εσωτερικά στο ίδιο αρχείο): Αν blocks διαφορετικών αρχείων (ή ακόμα και διαφορετικά blocks του ίδιου αρχείου) συμβεί να έχουν ακριβώς το ίδιο περιεχόμενο, δεν είναι απαραίτητο να αποθηκεύονται πολλαπλές φορές στο δίσκο.

Προκειμένου να αποφύγετε αλλαγές στον kernel (και τις συνακόλουθες επιβαρύνσεις χρόνου, δυσκολίες αποσφαλμάτωσης κλπ) το σύστημα αρχείων θα υλοποιηθεί ως σύστημα αρχείων επιπέδου χρήστη αξιοποιώντας την υποδομή FUSE (`https://github.com/libfuse/libfuse`).

Επίσης, θα χρειαστεί να υλοποιήσετε κατάλληλα scripts και αρχεία ελέγχου ώστε να επιβεβαιώσετε τη σωστή λειτουργία του συστήματος αρχείων που υλοποιήσατε.

1.1 Προαπαιτούμενα

1.1.1 Μελέτη

Πριν ξεκινήσετε την υλοποίηση θα πρέπει να έχετε διαβάσει το κεφάλαιο 13 από το βιβλίο ”Linux Kernel Development”, με έμφαση στα File Operations. Επίσης, θα πρέπει να έχετε μελετήσει το tutorial για τη χρήση του FUSE filesystem το οποίο θα βρείτε στο `https://www.cs.nmsu.edu/~pfeiffer/fuse-tutorial/` (και τοπικό αντίγραφο στη σελίδα του μαθήματος). Μελετήστε πολύ προσεκτικά το παράδειγμα που δίνεται στο tutorial (Big Brother File System - BBFS). Μπορείτε να χρησιμοποιήσετε το BBFS σαν αρχικό σημείο για την υλοποίησή σας, αξιοποιώντας ελεύθερα όποιο κομμάτι του κώδικα θέλετε.

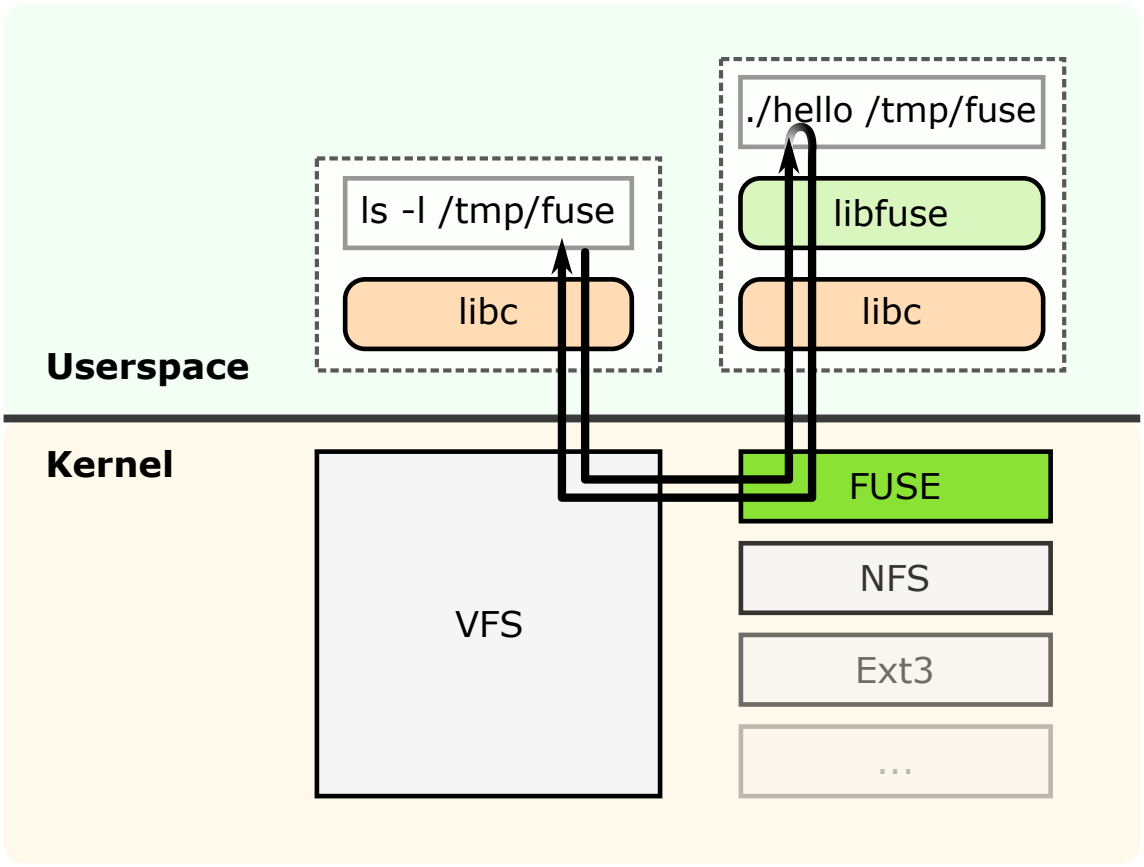
1.1.2 Προετοιμασία του συστήματος

Θα χρειαστεί να εγκαταστήσετε στο Linux VM που χρησιμοποιείτε για τις εργασίες κάποια πακέτα, με την παρακάτω ακολουθία εντολών:

```
sudo apt-get update
sudo apt-get install pkg-config libssl-dev libfuse-dev
```

1.1.3 Σύντομη εισαγωγή στο FUSE

Σε αρκετές περιπτώσεις θέλουμε να υλοποιήσουμε συστήματα αρχείων (ή ορθότερα εικονικά συστήματα αρχείων) στο επίπεδο χρήστη. Όμως αυτό ήταν — έως κάποιο χρονικό σημείο — αδύνατο, καθώς απαιτείται συνεργασία από τον πυρήνα του λειτουργικού για την πρόσβαση στο δίσκο. Αυτή την ανάγκη ήρθε να καλύψει το File System in USErspace (FUSE). Η βασική ιδέα του FUSE είναι η εισαγωγή στον πυρήνα ενός module το οποίο αλληλεπιδρά με τον πυρήνα (και πιο συγκεκριμένα με το Virtual File System - VFS) για λογαριασμό εφαρμογών χρήστη (χωρίς δικαιώματα διαχειριστή) και προσφέρει ένα API το οποίο μπορεί να αξιοποιηθεί από το επίπεδο χρήστη. Η εικόνα 1 απεικονίζει σχηματικά την αρχιτεκτονική και τη λογική λειτουργίας του FUSE.



Εικόνα 1: Αρχιτεκτονική και λογική λειτουργίας του FUSE (πηγή: Wikipedia).

Η λειτουργικότητα του user-level filesystem (έστω “hello world FS”) υλοποιείται από ένα εκτελέσιμο (έστω με όνομα *hello*). Το εκτελέσιμο τρέχει (πάνω δεξιά στην εικόνα) δημιουργώντας ένα mount point με όνομα */tmp/fuse*.

Έστω ότι ο χρήστης εκτελεί μία εντολή κάτω από αυτό το mount point (π.χ. *ls -l /tmp/fuse* - άπνω αριστερά στην εικόνα). Η εφαρμογή *ls* καλεί κάποιες συναρτήσεις της *libc* που υλοποιούν λειτουργικότητα του συστήματος αρχείων. Αυτές προκαλούν την κλήση των αντίστοιχων system calls (τα οποία υλοποιούνται στο *VFS*). Το *VFS*, διαπιστώνοντας ότι αυτά τα system calls αφορούν ένα user-level filesystem τα προωθεί στο FUSE module του πυρήνα. Αυτό με τη σειρά του τα προωθεί στο εκτελέσιμο που υλοποιεί το filesystem στο επίπεδο χρήστη, μετά από μεσολάβηση συναρτήσεων των βιβλιοθηκών *libc* και *libfuse*. Το εκτελέσιμο *hello* παράγει τα αποτελέσματα που ζήτησε η εφαρμογή *ls* και αυτά επιστρέφονται στην εφαρμογή (τελικά ως αποτέλεσμα της συνάρτησης της libc που είχε καλέσει η εφαρμογή) ακολουθώντας ακριβώς την αντίστροφη πορεία.

Για περισσότερες λεπτομέρειες πέρα από τη γενική εικόνα, διαβάστε προσεκτικά το tutorial που προαναφέρθηκε.

1.2 Προθεσμία και Τρόπος Παράδοσης

Η άσκηση θα πρέπει να παραδοθεί έως τα **μεσάνυχτα της Παρασκευής 31/5/2019**. Η προθεσμία είναι τελική και δεν πρόκειται να δοθεί καμία παράταση – συνολικά ή ατομικά – για κανένα λόγο. Ο χρόνος που σας δίνεται υπερεπαρκεί. Χρησιμοποιήστε τον ”σοφά”.

Η παράδοση θα γίνει στο e-class. Για τις λεπτομέρειες της παράδοσης δείτε την παράγραφο 3.

1.3 Κώδικας Ηθικής

Κάθε ομάδα θα πρέπει να εργαστεί ανεξάρτητα. Είναι δεκτό και θεμιτό διαφορετικές ομάδες να ανταλλάξουν απόψεις σε επίπεδο γενικής ιδέας ή αλγόριθμου. Απαγορεύεται όμως κάθε συζήτηση / ανταλλαγή κώδικα ή ψευδοκώδικα. Σημειώστε ότι οι ασκήσεις θα ελεγχθούν τόσο από αυτόματο σύστημα όσο και από εμάς για ομοιότητες μεταξύ των ομάδων αλλά και για ομοιότητες με τυχόν λύσεις που μπορεί να βρεθούν στο διαδίκτυο ή λύσεις που έχουν δοθεί σε προηγούμενα έτη. Σε περίπτωση αντιγραφής οι ασκήσεις (όλων των φάσεων) όλων των εμπλεκόμενων φετινών ομάδων μηδενίζονται χωρίς καμία περαιτέρω συζήτηση.

Όλα τα μέλη στο εσωτερικό κάθε ομάδας θα πρέπει να έχουν ισότιμη συμμετοχή στην ανάπτυξη κάθε φάσης. Διατηρούμε το δικαίωμα να επιβεβαιώσουμε αν αυτό τηρείται με προσωπικές συνεντεύξεις / προφορική εξέταση.

2 Ζητούμενα

Θα πρέπει να αξιοποιήσετε το FUSE ώστε να υλοποιήσετε ένα απλουστευτικό σύστημα αρχείων, το οποίο προσπαθεί να ελαχιστοποιήσει τον αριθμό των blocks που χρειάζονται στο δίσκο για την αποθήκευση ενός αριθμού από αρχεία εντοπίζοντας και επαναχρησιμοποιώντας τυχόν blocks με όμοιο περιεχόμενο.

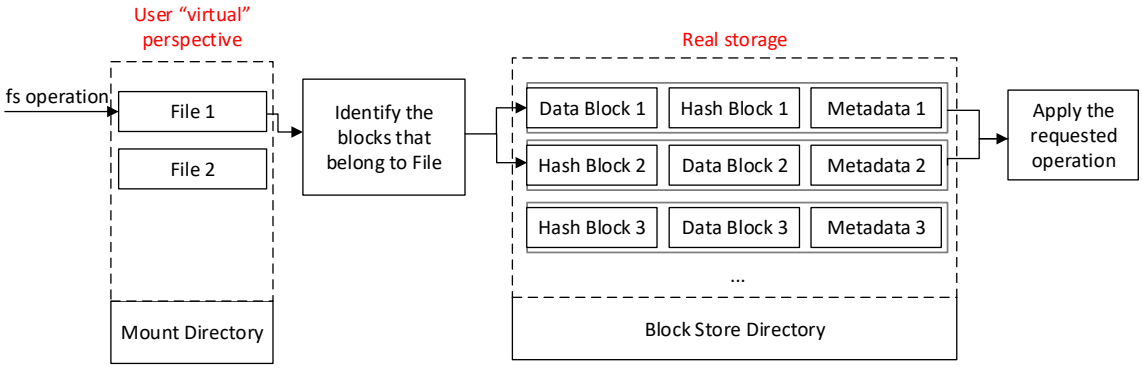
Τα αρχεία αποτελούνται, ως γνωστόν, από blocks και οι εφαρμογές εκτελούν αναγνώσεις και εγγραφές αυτών των blocks. Στην περίπτωση των εγγραφών, δε θα χρειαζόταν να γραφτεί ένα νέο block στο δίσκο, αν τυχόν ένα ήδη υπάρχον block στο δίσκο συμβεί αν έχει ίδιο περιεχόμενο. Προκειμένου να μην συγκρίνουμε πλήρη blocks (καθώς αυτό θα ήταν αργό), μπορούμε να διατηρούμε ένα hash για κάθε υπάρχον block στον δίσκο. Υπολογίζουμε το αντίστοιχο hash για το νέο block που ζητείται να γραφτεί και συγκρίνουμε με τα hashes των υπαρχόντων blocks. Εάν δύο hashes είναι ίσα, μπορείτε να θεωρήσετε ότι τα περιεχόμενα των αντίστοιχων blocks είναι ίσα. Με αυτό τον τρόπο γράφονται στο δίσκο αποκλειστικά μοναδικά (νέα) blocks. Φυσικά για τα αρχεία τα οποία επιχείρησαν να γράψουν blocks που είναι αντίγραφα προϋπαρχόντων θα πρέπει μετέπειτα αναγνώσεις να αποδίδουν το σωστό περιεχόμενο (άρα με κάποιον τρόπο το αρχείο θα πρέπει να αποκτά έναν “pointer” προς το προϋπάρχον block). Αυτή η τεχνική θα μειώσει σημαντικά τον απαιτούμενο αποθηκευτικό χώρο στο δίσκο, εφόσον υπάρχει σημαντική επικάλυψη στα περιεχόμενα των αρχείων.

Μπορείτε να επεκτείνετε τον κώδικα του Big Brother FileSystem (BBFS) που σας δίνεται ώστε να “συμπιέζετε” με τον τρόπο που συζητήθηκε νωρίτερα το χώρο αποθήκευσης που απαιτείται για τα αρχεία τα οποία εξυπηρετούνται από το BBFS.

2.1 Σχεδιαστικές αποφάσεις

Κατά τη σχεδίαση και υλοποίηση του συστήματος αρχείων σας θα χρειαστει να λάβετε κάποιες αποφάσεις. Για παράδειγμα, πώς αποθηκεύετε και διαχειρίζεστε τα blocks, τα περιεχόμενά τους και τα hashes τους; Μία επιλογή είναι να διατηρείτε στο δίσκο μία “αποθήκη blocks” όπου αποθηκεύονται τα περιεχόμενα των μοναδικών blocks και τα αντίστοιχα hashes. Νέες αιτήσεις εγγραφών ελέγχονται αν οδηγούν σε blocks που είναι όμοια με προϋπάρχοντα. Τα αρχεία που τελικά γράφονται στο δίσκο δεν είναι τίποτε άλλο παρά δείκτες προς μοναδικά blocks στην αποθήκη. Οι αιτήσεις ανάγνωσης θα πρέπει να ανακατασκευάζουν το αρχείο ακολουθώντας τους εν λόγω δείκτες.

Η λογική αυτής της αρχιτεκτονικής του συστήματος αρχείων παρουσιάζεται στην Εικόνα 2.



Εικόνα 2: Ενδεικτική αρχιτεκτονική του συστήματος αρχείων.

Αυτή δεν είναι η μοναδική σωστή σχεδιαστική απόφαση. Μπορείτε ελεύθερα να επιλέξετε το δικό σας τρόπο υλοποίησης.

2.1.1 Παραδοχές - απλουστεύσεις

Η υλοποίηση ενός πλήρως λειτουργικού συστήματος αρχείων είναι αρκετά επίπονη. Για το λόγο αυτό, στα πλαίσια αυτής της εργασίας μπορείτε — για τη δική σας διευκόλυνση — να κάνετε τις παρακάτω παραδοχές:

- Υποθέστε ότι όλες οι αιτήσεις ανάγνωσης και εγγραφής γίνονται σε αποστάσεις από την αρχή του αρχείου που είναι πολλαπλάσια των 4KB. Επιπλέον, μπορείτε να δουλέψετε με αρχεία που έχουν μέγεθος αποκλειστικά πολλαπλάσιο των 4KB.

- Αρκεί να υποστηρίξετε απλές αναγνώσεις και εγγραφές σε αρχεία, καθώς και δημιουργία και διαγραφή αρχείων. Γενικώς αρκεί η αντιγραφή ενός αρχείου στο σύστημα αρχείων σας και η ανάγνωσή του, καθώς και η μετέπειτα διαγραφή του να δουλεύουν σωστά. Δε χρειάζεται να υποστηρίζετε πιο πολύπλοκες ενέργειες (π.χ. truncation).

- Μπορείτε να υποθέσετε ότι το σύστημα αρχείων δε θα φιλοξενεί παραπάνω από 10 αρχεία, καθένα μικρότερο ή ίσο των 64KB.

- Αν τυχόν δύο blocks παράγουν το ίδιο hash, μπορείτε να υποθέσετε ότι είναι ίδια. Στην πραγματικότητα, με δεδομένο ότι ρεαλιστικά το hash αποτελείται από πολύ λιγότερα bits σε σχέση με το μέγεθος του block, περισσότερα από ένα διαφορετικά στιγμιότυπα περιεχομένου είναι

δυνατόν να οδηγήσουν στο ίδιο hash. Το φαινόμενο αυτό λέγεται hash collision. Στα πλαίσια της εργασίας θα το αγνοήσουμε.

- Μπορείτε να θεωρήσετε το filesystem volatile (να μην απαιτείται δηλαδή αν κάνετε unmount και μετέπειτα mount να βρείτε μέσα τα αρχεία που είχατε απόθηκεύσει πριν το unmount.

- Μπορείτε να θεωρήσετε ότι ένα μόνο πρόγραμμα (και native thread) προσπελαύνει το σύστημα αρχείων κάθε χρονική στιγμή. Δεν απαιτείται δηλαδή να ασχοληθείτε με ζητήματα thread safety της υλοποίησης του filesystem.

Γενικά, αν αξιοποιήσετε όλες τις παραπάνω παραδοχές / απλουστεύσεις περιμένουμε ότι θα χρειαστεί να υλοποιήσετε συναρτήσεις για τα ακόλουθα: *readir*, *opendir*, *getattr*, *open*, *close*, *read*, *write*, *unlink*, καθώς και τις συναρτήσεις για initialization / destruction του συστήματος αρχείων.

Εάν επιλεξετε να μην αξιοποιήσετε (κάποιες από) τις παραπάνω απλουστεύσεις / παραδοχές είναι δυνατό να κερδίσετε επιπλέον βαθμολογικό bonus.

2.2 Βοήθεια σε ζητήματα υλοποίησης

Δώστε προσοχή στα παρακάτω σημεία:

- Μην κάνετε το λάθος να κάνετε μέσα από την υλοποίηση του συστήματος αρχείων σας πράξεις αρχείων στο directory κάτω από το οποίο έχει γίνει mount το σύστημα αρχείων σας (αντί του ορθού να τις κάνετε στο βοηθητικό directory στο οποίο αποθηκεύετε blocks, hashes, metadata αρχείων κλπ). Σε αντίθετη περίπτωση θα προκαλέσετε μία ατέρμονη σειρά κλήσεων.

  Έστω π.χ. ότι το σύστημα αρχείων σας έχει γίνει mount κάτω από το */home/myaccount/myfs*, ενώ το directory που χρησιμοποιείται ως "αποθήκη" είναι το */home/myaccount/storage*. Έστω ότι θέλετε να ανοίξετε το αρχείο */home/myaccount/myfs/myfile.c*. Θυμηθείτε ότι κατά την υλοποίηση της *open()* σας δε θα πρέπει να κάνετε οποιαδήποτε ενέργεια (*read(), write(), open()* κλπ) στο */home/myaccount/myfs*. Όποιες ενέργειες γίνουν από τον κώδικα που υλοποιεί το filesystem σας, θα πρέπει να αφορούν αρχεία κάτω από το */home/myaccount/storage*.

- Δώστε προσοχή στην υλοποίηση της *getattr()*, ειδικά για το μέγεθος του αρχείου που η εν λόγω συνάρτηση επιστρέφει. Θα πρέπει να επιστρέφεται όχι το μέγεθος των μεταδεδομένων (καθώς το αρχείο στην υλοποίησή σας στην ουσία αποτελείται μόνο από μεταδεδομένα -

pointers στην αποθήκη blocks) αλλά το μέγεθος που θα είχε το αρχείο (τα δεδομένα του) αν ήταν αποθηκευμένο με συμβατικό τρόπο.

Το παραπάνω είναι ιδιαίτερα σημαντικό, καθώς το μέγεθος του αρχείου χρησιμοποιείται εμμέσως και στις *read()* και *write()*, οι οποίες θα έχουν μη αναμενόμενη συμπεριφορά αν δε φροντίσετε να επιστρέφετε το σωστό μέγεθος.

- Οι συναρτήσεις που επιστρέφουν τα περιεχόμενα ενός καταλόγου (και χρησιμοποιούνται π.χ. από την ls) θα πρέπει να αποκρύπτουν βοηθητικά αρχεία (π.χ. blocks, hashes) τα οποία υπάρχουν στον κατάλογο που χρησιμοποιείτε για την υλοποίηση της "αποθήκης".

- Αν επιλέξετε να υλοποιήσετε volatile σύστημα αρχείων, φροντίστε πριν το mount ή μετά το unmount να καθαρίζετε όλα τα αρχεία από την "αποθήκη". Αντίστροφα, αν επιλέξετε να υλοποιήσετε non-volatile σύστημα αρχείων θα πρέπει κατά το mount να "σκανάρετε" την "αποθήκη" και να αρχικοποιείτε κατάλληλα τις εσωτερικές δομές του κώδικα του συστήματος αρχείων.

- Για τον υπολογισμό των hashes μπορείτε να χρησιμοποιήσετε τη συνάρτηση *SHA1* από τη βιβλιοθήξη OpenSSL. Στην περίπτωση που θέλετε να κάνετε compile τα sources του bbfs με υποστήριξη για κλήσεις της βιβλιοθήκης openssl, βεβαιωθείτε πως έχετε τρέξει την εντολή *./configure* στο root των αρχείων του bbfs και στην συνέχεια αντικαταστήσε τα περιεχόμενα του *./src/Makefile* με τα παρακάτω

BBFS Makefile with openssl

```
all:
        gcc -g -Wall bbfs.c log.c \
                `pkg-config fuse --cflags --libs` \
                `pkg-config libssl --cflags --libs` -o bbfs

clean:
        rm -rf ./bbfs
```

2.3 Έλεγχος λειτουργικότητας / ορθότητας

Θα πρέπει στο παραδοτέο σας (κώδικα και αναφορά) να περιλαμβάνεται **τουλάχιστον ένα** πείραμα το οποίο σχεδιάσατε και εκτελέσατε για να ελέγξετε τη λειτουργικότητα και την ορθότητα της

υλοποίησής σας. Για παράδειγμα:

1. Κατασκευάστε δύο αρχεία με πολύ παρόμοιο περιεχόμενο και αντιγράψτε τα το ένα μετά το άλλο μέσα στον κατάλογο στον οποίο έχετε κάνει mount το σύστημα αρχείων σας.

2. Το σύστημα αρχείων θα πρέπει να αναγνωρίσει κοινά blocks και να αποθηκεύσει μόνο ένα αντίγραφο από κάθε κοινό block στο χώρο στο δίσκο που υλοποιεί την αποθήκη blocks. Κατά συνέπεια, ο χώρος που καταλαμβάνει η αποθήκη blocks θα πρέπει να είναι μικρότερος από το συνολικό χώρο που καταλάμβαναν τα αντίστοιχα αρχεία σε ένα συμβατικό σύστημα αρχείων. Επιπλέον, τα περιεχόμενα των αρχείων στον βοηθητικό κατάλογο που “εξυπηρετεί” το σύστημα αρχείων (προσοχή, όχι αυτόν που κάνατε mount και βλέπει ο χρήστης) θα είναι διαφορετικά σε σχέση με των αρχείων στο συμβατικό σύστημα αρχείων, καθώς το περιεχόμενο των αρχείων στον 1ο κατάλογο δεν είναι τίποτε άλλο παρά pointers στην αποθήκη blocks.

3. Όταν τα αρχεία που γράφτηκαν στον κατάλογο στον οποίο κάναμε mount το σύστημα αρχείων διαβαστούν (από τον ίδιο κατάλογο) θα πρέπει να βλέπουμε το κανονικό τους περιεχόμενο, όπως αυτό θα φαινόταν διαβάζοντάς τα από ένα συμβατικό σύστημα αρχείων.

3 Πακετάρισμα των Αλλαγών – Δημιουργία του προς Υποβολή Συνημμένου Αρχείου

Το τελευταίο βήμα είναι να πακετάρετε όλα τα αρχεία που πρέπει να μας στείλετε σε ένα αρχείο. Πηγαίνετε στο home directory του λογαριασμού σας. Φτιάξτε εκεί με την εντολή mkdir έναν κατάλογο με όνομα project_4_AEM1_AEM2_AEM3. Για παράδειγμα, η ομάδα με μέλη τους φοιτητές με ΑΕΜ 1234 2345 3456 θα πρέπει να δώσει την εντολή mkdir project_4_1234_2345_3456.

Φτιάξτε μέσα σε αυτόν τον κατάλογο 3 υποκαταλόγους: (α) filesystem, στον οποίο θα βάλετε όλα τα αρχεία με την υλοποίηση του filesystem σας συμπεριλαμβανομένου του Makefile, (β) experiments, στον οποίο θα βάλετε τα αρχεία που χρησιμοποιήσατε στην αξιολόγηση, τυχόν scripts κλπ, (γ) report, στον οποίο θα βάλετε την αναφορά σας. Η αναφορά θα πρέπει να περιλαμβάνει τα ονόματα, ΑΕΜ και e-mail των μελών της ομάδας, συζήτηση για την αρχιτεκτονική του συστήματος αρχείων και τις σχεδιαστικές σας αποφάσεις, κατάλογο των συναρτήσεων του BBFS που χρειάστηκε να αλλάξετε, περιγραφή της πειραματικής σας αξιολόγησης και οδηγίες για την εκτέλεση της πειραματικής αξιολόγησης. Μπορείτε να συμπεριλάβετε στο αρχείο και οτιδήποτε άλλο θέλετε να

έχουμε υπόψη μας κατά τη διόρθωση. Το αρχείο μπορεί να είναι απλό αρχείο κειμένου (προσέξτε το encoding να είναι UTF-8 (unicode)), ή — αν προτιμάτε — αρχείο pdf.

Ακολούθως, δώστε την εντολή

tar -cvf project_4_AEM1_AEM2_AEM3.tar project_4_AEM1_AEM2_AEM3

Για παράδειγμα, η ομάδα με μέλη 1234 2345 3456 θα δώσει την εντολή

tar -cvf project_4_1234_2345_3456.tar project_4_1234_2345_3456

Με την εντολή tar θα πακεταριστούν τα περιεχόμενα του καταλόγου σε ένα αρχείο με κατάληξη .tar.

Τέλος, δώστε την εντολή bzip2 project_4_AEM1_AEM2_AEM3.tar. Θα δημιουργηθεί ένα αρχείο με κατάληξη .bz2, το οποίο περιέχει συμπιεσμένο το αρχείο με κατάληξη .tar. Για παράδειγμα, η ομάδα 1234 2345 3456 θα δώσει την εντολή bzip2 project_4_1234_2345_3456.tar. **Το αρχείο με κατάληξη .bz2 είναι αυτό που θα πρέπει να επισυνάψετε κατά την παράδοση της άσκησης στο e-class**.