ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
http://www.cslab.ece.ntua.gr

Εργαστήριο Λειτουργικών Συστημάτων

7ο εξάμηνο, Ακαδημαϊκή περίοδος 2024–2025

Οδηγός εργαστηριακής άσκησης
Συστήματα Αρχείων στο Linux

Εργαστήριο Υπολογιστικών Συστημάτων Ε.Μ.Π.
os-lab@lists.cslab.ece.ntua.gr

Δεκέμβριος 2024

Περιεχόμενα

Επιμέλεια: Δ. Σιακαβάρας, Φ. Τόφαλος

1 Εισαγωγή

Τα συστήματα αρχείων είναι απο τα σημαντικότερα κομμάτια ενός ΛΣ και καθορίζουν την οργάνωση των αρχείων και τον τρόπο με τον οποίο αυτά αποθηκεύονται σε συσκευές μόνιμης αποθήκευσης, π.χ., σκληρούς δίσκους, συσκευές αποθήκευσης USB, κλπ. Υπάρχουν πάρα πολλά διαφορετικά συστήματα αρχείων και κάθε σύγχρονο ΛΣ πρέπει να μπορεί να υποστηρίξει μεγάλο κομμάτι αυτών. Σκοπός της παρούσας εργαστηριακής άσκησης είναι η εξοικείωσή σας με το κομμάτι του πυρήνα του Linux που σχετίζεται με την διαχείριση των συστημάτων αρχείων (γνωστό ως VFS, Virtual File System ή Virtual Filesystem Switch) και η κατανόηση ενός συγκεκριμένου συστήματος αρχείων, του ext2, που ανήκει στην οικογένεια των ext συστημάτων αρχείων που χρησιμοποιούνται ευρέως στο Linux.

Όλες οι πληροφορίες που περιέχονται στον οδηγό και αφορούν σε κομμάτια του πυρήνα του Linux είναι ακριβή για την έκδοση 6.11 του πυρήνα του Linux. Αρκετά απο τα κομμάτια που αφορούν το VFS προέρχονται απο την 3η έκδοση του βιβλίου των Bovet και Cesati με τίτλο "Understanding the Linux Kernel" [1]. Γενικά σας προτείνουμε και προτρέπουμε να αναζητήσετε το βιβλίο και να διαβάσετε τα κεφάλαια 12 (The Virtual Filesystem), 15 (The Page Cache), 16 (Accessing Files) και 18 (The Ext2 and Ext3 Filesystems) που σίγουρα θα σας βοηθήσουν να κατανοήσετε σε βάθος τη λειτουργία του Linux όσον αφορά τα συστήματα αρχείων.

2 Κατηγορίες συστημάτων αρχείων

Τα συστήματα αρχείων που υποστηρίζονται στο Linux χωρίζονται σε τρεις κατηγορίες:

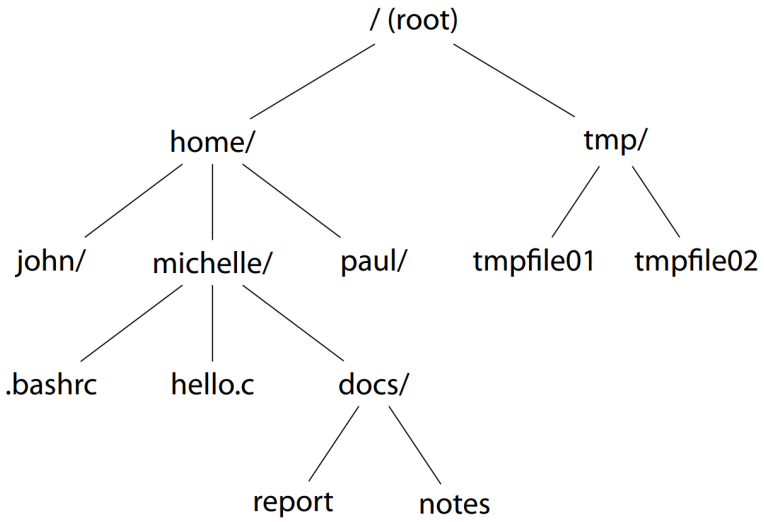
- *Συστήματα αρχείων σε συσκευές μόνιμης αποθήκευσης:* Συστήματα αρχείων τα οποία αναλαμβάνουν την οργάνωση των αρχείων μέσα σε κάποια συσκευή μόνιμης αποθήκευσης, όπως ένας σκληρός δίσκος, μία δισκέττα, μία συσκευή flash USB, ένα CD-ROM, κλπ. Μερικά απο τα πιο γνωστά τέτοια συστήματα αρχείων είναι τα ext2, ext3, ext4 στο Linux και NTFS, FAT σε Windows (με υποστήριξη και στο Linux).

- *Δικτυακά συστήματα αρχείων:* Συστήματα αρχείων τα οποία παρέχουν πρόσβαση μέσω δικτύου σε αρχεία που βρίσκονται σε απομακρυσμένους υπολογιστές. Κάποια πολύ γνωστά δικτυακά συστήματα αρχείων είναι τα NFS (Network File System), sshfs στο Linux και το CIFS (Common Internet File System) σε Windows.

- *Ειδικά συστήματα αρχείων:* Συστήματα αρχείων τα οποία δε χρησιμοποιούν ούτε κάποια συσκευή αποθήκευσης ούτε το δίκτυο. Χρησιμοποιούνται για την ανταλλαγή διαφόρων ειδών πληροφορίας ανάμεσα στον πυρήνα του Linux και τον χώρο χρήστη. Κλασσικά παραδείγματα τέτοιων συστημάτων αρχείων είναι το `procfs` και το `devtmpfs`. Το `procfs` συνήθως προσαρτάται στο `/proc` και εξάγει ειδικά αρχεία μέσω των οποίων ο χρήστης μπορεί να πάρει πληροφορίες απο τον πυρήνα (π.χ., το /proc/meminfo δίνει πληροφορίες για τη χρήση της μνήμης RAM απο το ΛΣ). Το `devtmpfs` συνήθως προσαρτάται στο `/dev` και περιλαμβάνει τα αρχεία ειδικών συσκευών για τις περιφερειακές συσκευές του μηχανήματος.

3 Ιεραρχία αρχείων στο Linux

Η ιεραρχία αρχείων στο Linux ακολουθεί δενδρική δομή και η ρίζα του δέντρου είναι το γνωστό `root filesystem`. Οι ενδιάμεσοι κόμβοι του δέντρου είναι κατάλογοι οι οποίοι περιέχουν αρχεία ή/και άλλους καταλόγους. Τα φύλλα του δέντρου είναι είτε κανονικά αρχεία είτε άλλου τύπου αρχεία όπως pipes, αρχεία συσκευών (π.χ., /dev/lunix0-bat), σύνδεσμοι, κλπ. Στο Σχήμα 1 δίνεται ένα παράδειγμα μιας ιεραρχικής δομής αρχείων σε ΛΣ Linux.



Σχήμα 1: Παράδειγμα δενδρικής ιεραρχικής δομής αρχείων στο Linux.

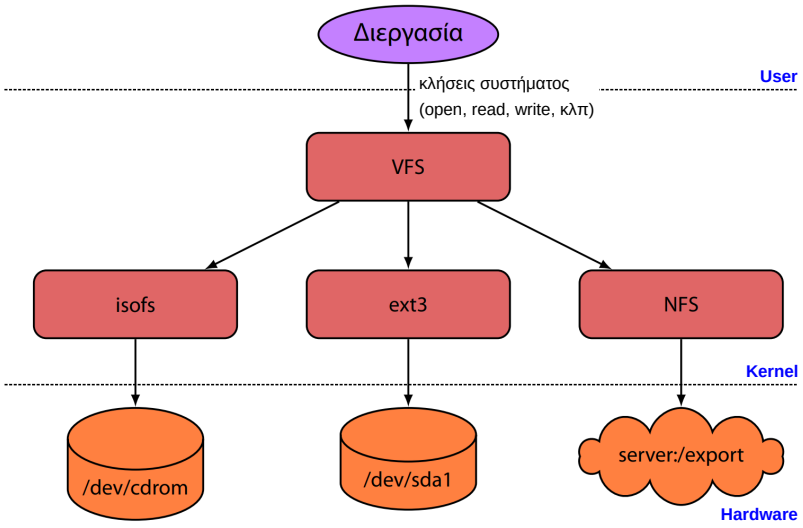
Για την αναφορά σε κάποιο αρχείο (ή κατάλογο) οι χρήστες χρησιμοποιούν το αντίστοιχο μονοπάτι (path) στο δέντρο. Τα μονοπάτια μπορεί να είναι είτε απόλυτα είτε

σχετικά. Τα απόλυτα μονοπάτια ξεκινάνε με τον χαρακτήρα / και η αφετηρία τους είναι πάντα η ρίζα του δέντρου. Τα σχετικά μονοπάτια **δεν** ξεκινάνε με τον χαρακτήρα / και η αφετηρία τους είναι ο τρέχων κατάλογος της διεργασίας η οποία έκανε την κλήση συστήματος που αφορά το συγκεκριμένο μονοπάτι. Ο τρέχων κατάλογος μιας διεργασίας αλλάζει μέσω της κλήσης συστήματος `chdir()`.

Για παράδειγμα, δεδομένου της δενδρικής δομής του Σχήματος 1, και αν θεωρήσουμε μια διεργασία με τρέχων κατάλογο το `/home/michelle/`, τα δύο παρακάτω μονοπάτια αναφέρονται στο ίδιο αρχείο:

- `/home/michelle/hello.c`: απόλυτο μονοπάτι.
- `hello.c`: σχετικό μονοπάτι.

4 Virtual FileSystem (VFS)



Σχήμα 2: Η διεπαφή του VFS στο Linux

Το Linux είναι ένα ΛΣ που υποστηρίζει πολλαπλά συστήματα αρχείων μέσω μιας διεπαφής που ονομάζεται VFS (Virtual File System ή αλλιώς Virtual Filesystem Switch). Όπως γνωρίζετε απο τις προηγούμενες ασκήσεις του μαθήματος, γενικά η διεπαφή του Linux για πρόσβαση στον πυρήνα (και κατ' επέκταση στο υλικό του υπολογιστή) είναι οι κλήσεις συστήματος (system calls). Το VFS είναι το κομμάτι εκείνο που αναλαμβάνει την εκτέλεση των κλήσεων συστήματος που αφορούν στη διαχείριση αρχείων. Είναι δηλαδή στην ουσία ένα σύνολο συναρτήσεων που βρίσκεται ανάμεσα

απο τον χρήστη και τα συστήματα αρχείων και μεταφράζει κλήσεις συστήματος σε αντίστοιχες κλήσεις συναρτήσεων για το εκάστοτε σύστημα αρχείων.

4.1 Καταχώρηση/Διαγραφή συστημάτων αρχείων στον πυρήνα

Το VFS υποστηρίζει πολλαπλά ΣΑ και προσφέρει τη δυνατότητα για προσθήκη νέων, ακόμα και δυναμικά χρησιμοποιώντας κάποιο linux module. Μέσω του ειδικού αρχείου `/proc/filesystems` μπορούμε να δούμε τα ΣΑ που υποστηρίζονται στο σύστημά μας:

```
$ cat /proc/filesystems
nodev   sysfs
nodev   tmpfs
nodev   proc
...
nodev   pipefs
nodev   devpts
        ext3
        ext2
        ext4
        vfat
...
```

Το VFS υποστηρίζει την δυναμική καταχώρηση/διαγραφή συστημάτων αρχείων στον πυρήνα. Για το σκοπό αυτό χρησιμοποιούνται οι δύο παρακάτω συναρτήσεις:

- `int register_filesystem(struct file_system_type *);`
- `int unregister_filesystem(struct file_system_type *);`

Η `register_filesystem` καλείται είτε κατα τη διαδικασία εκκίνησης του πυρήνα (στην περίπτωση που ο κώδικας του συστήματος αρχείων είναι μέρος του κώδικα του πυρήνα) είτε κατα την εισαγωγή του αντίστοιχου module στον πυρήνα. Η δομή `file_system_type` που δίνεται σαν όρισμα στις παραπάνω συναρτήσεις έχει τα παρακάτω πεδία (εδώ αναφέρουμε μόνο τα πεδία που θα μας απασχολήσουν στα πλαίσια της άσκησης):

- `const char *name;`

  Το όνομα του συστήματος αρχείων το οποίο θα εμφανίζεται και στο `/proc/filesystems`.

- `int fs_flags;`

  Μία μάσκα απο bits που χρησιμοποιείται για να οριστούν διάφορες παράμετροι του συστήματος αρχείων.

- `struct dentry *mount(struct file_system_type *, int flags, const char *dev_name, void *data);`

  Αυτή είναι η μέθοδος που καλείται όταν το VFS θέλει να προσαρτήσει κάποιο σύστημα αρχείων που χρησιμοποιεί αυτό το `file_system_type`. Ο ρόλος της είναι να δεσμεύσει τη μνήμη που χρειάζεται για το `super_block` και να κάνει τις απαραίτητες αρχικοποιήσεις. Η παράμετρος `flags` καθορίζει τις παραμέτρους με τις οποίες θα προσαρτηθεί το νέο σύστημα αρχείων. Η παράμετρος `dev_name` είναι το όνομα της συσκευής στην οποία περιέχεται το σύστημα αρχείων (μπορεί να είναι NULL στις περιπτώσεις ειδικών συστημάτων αρχείων). Τέλος, η παράμετρος `data` περιέχει επιπλέον επιλογές προσάρτησης που μπορεί να δώσει ο χρήστης στην κλήση συστήματος `mount`. Για τα συστήματα αρχείων σε συσκευές block ο πυρήνας παρέχει τη γενική συνάρτηση `mount_bdev` η οποία αναλαμβάνει τη γενική δέσμευση μνήμης και αρχικοποίηση και αφήνει στην ευθύνη του κώδικα του συστήματος αρχείων μόνο το να "γεμίσει" το `super_block` με τα κατάλληλα δεδομένα. Για να γίνει αυτό η `mount_bdev` παίρνει σαν όρισμα ένα δείκτη σε μια μέθοδο `fill_super` η οποία υλοποιείται απο τον οδηγό του συστήματος αρχείων.

- `void kill_sb(struct super_block *);`

  Αυτή είναι η μέθοδος που καλείται όταν το VFS θέλει να αποπροσαρτήσει κάποιο σύστημα αρχείων που χρησιμοποιεί αυτό το `file_system_type`. Ο ρόλος της είναι να αποδεσμεύσει τη μνήμη που σχετίζεται με το `super_block` του συγκεκριμένου συστήματος αρχείων και να διαγράψει οτιδήποτε άλλο σχετικό. Για συστήματα αρχείων που χρησιμοποιούν συσκευές block παρέχεται απο τον πυρήνα μία γενική συνάρτηση `kill_block_super` που μπορεί να χρησιμοποιηθεί απο συστήματα αρχείων που δε χρειάζεται να κάνουν κάποια επιπλέον ενέργεια.

4.2 Προσάρτηση/Αποπροσάρτηση συστημάτων αρχείων

Αφού έχουμε καταχωρήσει τον τύπο του συστήματος αρχείων μας, μπορούμε να προσαρτήσουμε/αποπροσαρτήσουμε δίσκους που περιέχουν το σύστημα αρχείων μας. Μέσω του ειδικού αρχείου `/proc/mounts` μπορούμε να δούμε τα ΣΑ που είναι προσαρτημένα στο σύστημά μας:

```
$ cat /proc/mounts
```

```
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
udev on /dev type devtmpfs (rw,nosuid,...)
devpts on /dev/pts type devpts (rw,nosuid,noexec,...)
tmpfs on /run type tmpfs (rw,nosuid,nodev,noexec,relatime,...)
...
/dev/vda1 on / type ext4 (rw,relatime,errors=remount-ro)
...
```

Για την προσάρτηση ενός συστήματος αρχείου παρέχεται η κλήση συστήματος `mount` και για την αποπροσάρτηση η `umount` (man 2 mount, man 2 umount). Κατα την εκτέλεση της `mount` το VFS καλεί την αντίστοιχη συνάρτηση που δεικτοδοτείται απο το πεδίο `.mount` του κατάλληλου `file_system_type` και έτσι αποκτά πρόσβαση στο inode της ρίζας (root) του συγκεκριμένου συστήματος αρχείων. Παρακάτω εξηγούμε τις βασικές δομές του VFS, συμπεριλαμβανομένου του inode.

4.3 Οι βασικές δομές του VFS

4.3.1 Η δομή *super_block*

Η δομή super_block (`struct super_block`) περιλαμβάνει όλες τις πληροφορίες για κάθε σύστημα αρχείων που έχει προσαρτηθεί στη κεντρική δενδρική δομή αρχείων. Πέρα απο βασικές πληροφορίες όπως για παράδειγμα η συσκευή στην οποία αναφέρεται το σύστημα αρχείων, το μέγεθος του block που χρησιμοποιεί, κλπ, η δομή περιλαμβάνει το πεδίο *s_op* που είναι δείκτης σε μία δομή *super_operations*. Η συγκεκριμένη δομή περιέχει δείκτες σε συναρτήσεις οι οποίες καλούνται για να κάνουν ενέργειες που αφορούν τη δομή super_block. Το κάθε σύστημα αρχείων θέτει αυτό το δείκτη σε μία δομή όπου η κάθε συνάρτηση/μέθοδος γνωρίζει τι ενέργειες πρέπει να κάνει. Παρακάτω είναι οι συναρτήσεις/μέθοδοι τις οποίες θα δούμε στα πλαίσια της άσκησης:

- `struct inode *alloc_inode(struct super_block *);`

  Αυτή η μέθοδος καλείται όταν το VFS χρειάζεται να δεσμεύσει μνήμη για μία δομή inode. Στα περισσότερα ΣΑ δεσμεύεται μια μεγαλύτερη δομή η οποία περιέχει μέσα της το struct inode, το οποίο και επιστρέφεται απο τη μέθοδο.

- `void *free_inode(struct inode *);`

  Αυτή η μέθοδος καλείται όταν το VFS χρειάζεται να αποδεσμεύσει τη μνήμη ενός inode.

- `int write_inode(struct inode *, struct writeback_control *);`

  Δίνεται ως παράμετρος η δομή inode του VFS, και η υλοποίηση της συνάρτησης στο εκάστοτε σύστημα αρχείων αναλαμβάνει να αντιγράψει της πληροφορίες της παραμέτρου στην δική του επέκταση του inode (π.χ. `struct ext2_inode`).

- `void evict_inode(struct inode *);`

  Καλείται όταν πλέον σταματήσουν να υπάρχουν αναφορές στο αρχείο. Στο ext2, αν υπάρχει κάποιο hardlink προς το αρχείο, τότε απλά επιστρέφονται οι σελίδες του από την μνήμη. Διαφορετικά, γίνεται διαγραφή του αρχείου (αφαίρεση δομής inode, απελευθέρωση θέσεων στα bitmaps κλπ).

  Σε παλιότερες εκδόσεις του πυρήνα οι αντίστοιχες μέθοδοι ονομάζονταν `delete_inode` και `clear_inode`.

- `void put_super(struct super_block *);`

  Απελευθερώνει από την μνήμη δομές που σχετίζονται με το superblock, ενημερώνοντας το κατάλληλα. Καλείται κατά την απόπροσάρτηση του συστήματος αρχείων.

- `int sync_fs(struct super_block *sb, int);`

  Ενημέρωση δομών δεδομένων του συστήματος αρχείων στον δίσκο. Δεν μας αφορά τόσο στο ext2, καθώς δεν είναι journaling filesystem: δεν κρατάει δηλαδή λογαριασμό για τις αλλαγές στα μεταδεδομένα που δεν έχουν γραφτεί ακόμα στο δίσκο, ώστε να μπορεί να ανακάμψει ύστερα από απρόσμενη διακοπή λειτουργίας.

- `int statfs(struct dentry *, struct kstatfs *);`

  Καλείται μέσω της κλήσης συστήματος `statfs` και γεμίζει τη δομή `kstatfs` με πληροφορίες και στατιστικά σχετικά με το ΣΑ, όπως για παράδειγμα τον τύπο του ΣΑ, το μέγεθος του block, τον αριθμό των διαθέσιμων blocks, κλπ.

- `int remount_fs(struct super_block *, int *, char *);`

  Καλείται μέσω της κλήσης συστήματος `mount` (με χρήση της σημαίας `MS_REMOUNT`) και χρησιμοποιείται για την αλλαγή παραμέτρων ενός ήδη προσαρτημένου ΣΑ.

4.3.2 Η δομή *inode*

Η δομή `inode` (struct inode) αναπαριστά ένα ”αντικείμενο” μέσα στο σύστημα αρχείων. Με τον όρο ”αντικείμενο” εδώ εννούμε κάθε οντότητα που καταλαμβάνει

χώρο μέσα στο σύστημα αρχείων, δηλαδή, κανονικό αρχείο δεδομένων (regular file), κατάλογος (directory), συμβολικός σύνδεσμος (symbolic ή soft link) ή ειδικό αρχείο (FIFO, block, character, socket). Στα πλαίσια της άσκησης θα ασχοληθούμε μόνο με κανονικά αρχεία, καταλόγους και συμβολικούς συνδέσμους. Τα βασικά πεδία της δομής `inode` που θα μας απασχολήσουν είναι:

- `umode_t i_mode;`

  Τύπος αρχείου (character device, block device, regular file κλπ.) και δικαιώματα πρόσβασης (read, write, execute ως προς owner, group, others). Μπορείτε να δείτε για παράδειγμα αυτόν τον πίνακα από το documentation του ext4.

- `kuid_t i_uid;`

  Αναγνωριστικό χρήστη.

- `kgid_t i_gid;`

  Αναγνωριστικό group.

- `const struct inode_operations *i_op;`

  Ο δείκτης σε δομή με συλλογή (δεικτών) συναρτήσεων για λειτουργίες πάνω σε inode. Μέσω αυτής της δομής, μεταξύ άλλων, μπορεί το VFS να αναφέρεται στην υλοποίηση του εκάστοτε συστήματος αρχείου.

- `struct super_block *i_sb;`

  Δείκτης στο superblock με το οποίο συσχετίζεται το αρχείο.

- `struct address_space *i_mapping;`

  Δείκτης προς δομή ενδεικτική των σελίδων του inode στην page cache, και με ποιο τρόπο οφείλει να τις διαχειριστεί (μέσω της δομής `struct address_space_operations`, η οποία επίσης περιέχει δείκτες προς συναρτήσεις που υλοποιεί κάθε σύστημα αρχείων ξεχωριστά).

- `const unsigned int i_nlink;`

  Αριθμός από hard links προς το inode.

- `atomic_t i_count`

  Usage counter για το inode. Δείχνει κοινώς πόσες διεργασίες χρησιμοποιούν το αρχείο αυτό μια δεδομένη στιγμή.

  Σε αντίστοιχη πρόκληση του riddle, μπορεί κάποιοι να αξιοποιήσατε το γεγονός ότι κάποιο αρχείο δεν μπορούσε να διαγραφεί πλήρως με το `i_count` να είναι μη μηδενικό, ακόμα και αν το `i_nlink` ήταν.

- `unsigned long i_ino;`

  Αναγνωριστικό του inode. Μπορείτε να το λάβετε μέσω της εντολής `ls -i <file>`.

- `const struct file_operations *i_fop;`

  Δείκτης σε δομή file_operations, η οποία έχει δείκτες συναρτήσεων που υλοποιούν βασικές λειτουργίες πάνω σε αρχεία, πχ. `read`, `write`, `ioctl` κλπ. Την ίδια δομή έχετε συναντήσει και στην άσκηση του οδηγού Lunix.

Στην συνέχεια φαίνονται οι μέθοδοι για τη διαχείριση inodes που θα μας απασχολήσουν και ορίζονται στη δομή `inode_operations`. Για κάθε μέθοδο θεωρούμε δεδομένο ότι έχουν κατασκευαστεί προηγουμένως τα αντίστοιχα dentries (παρακάτω στον οδηγό μπορείτε να βρείτε πληροφορίες σχετικά με το τι είναι το VFS `dentry`).

- `int create(struct inode *,struct dentry *, umode_t, bool);`
- ```
  struct dentry *lookup(struct inode *,struct dentry *,
                        unsigned int);
  ```
- `int link(struct dentry *,struct inode *,struct dentry *);`
- `int unlink(struct inode *,struct dentry *);`
- `int symlink(struct inode *,struct dentry *,const char *);`
- `int mkdir(struct inode *,struct dentry *,umode_t);`
- `int rmdir(struct inode *,struct dentry *);`
- `int mknod(struct inode *,struct dentry *,umode_t,dev_t);`
- ```
  int rename(struct inode *, struct dentry *, struct inode *,
             struct dentry *, unsigned int);
  ```

4.3.3 Η δομή file

Η δομή file (`struct file`) αναπαριστά ένα ανοικτό αρχείο κάποιας διεργασίας. Δημιουργείται όταν η διεργασία ανοίγει το αρχείο, και απελευθερώνεται όταν εκμηδενιστούν οι αναφορές της δομής (οι διεργασίες παιδιά που δημιουργεί μια διεργασία μοιράζονται τις δομές file των ανοικτών αρχείων της, οπότε μπορούμε να έχουμε πάνω από μία αναφορά).

Η δομή λαμβάνει πολλές πληροφορίες από το inode, αλλά σε αντίθεση με αυτό, υπάρχει μόνο στην μνήμη και όχι στο μέσο αποθήκευσης. Διαθέτει επιπλέον το πεδίο `loff_t f_pos` που αντιστοιχεί στο offset του ανοικτού αρχείου. Για αυτό και διαφορετικές διεργασίες έχουν διαφορετικά offset, αλλά νήματα που κληρονόμησαν ένα ανοικτό αρχείο έχουν κοινό offset.

4.3.4 Η δομή *dentry*

Η δομή `dentry` (struct dentry) αντιστοιχεί σε κόμβους στα μονοπάτια της δενδρικής δομής αρχείων που παρουσιάζει το Linux στον χρήστη. Συγκεκριμένα, δημιουργούνται τα κατάλληλα dentry objects των κόμβων του μονοπατιού κατά την ανάγνωση του αντίστοιχου directory entry (pathname resolution) από το σύστημα αρχείων όπως είναι αποθηκευμένο στον δίσκο. Τα παραγόμενα αντικείμενα τύπου dentry αποθηκεύονται προσωρινά (cache dentry) για να επιταχυνθεί όποια μελλοντική τους πρόσβαση.

Όταν για παράδειγμα αναφερόμαστε στο μονοπάτι `/usr/local/` (γιατί θέλουμε, για παράδειγμα, πρόσβαση στο αρχείο `/usr/local/bin`), δημιουργούνται τρία dentries (τουλάχιστον όποια δεν υπάρχουν ήδη): ένα για το root directory, και δύο για τα directories `usr` και `local`.

Συνοπτικά λοιπόν, το dentry αποτελεί αναπαράσταση ενός directory στην μνήμη, και βοηθάει να θυμόμαστε επιλύσεις ονομάτων καταλόγων που έχουμε πραγματοποιήσει πρόσφατα. Εκτός αυτού, αναφέρεται παρενθετικά ότι ένα dentry μπορεί να έχει ρόλο απομνημόνευσης αποτυχημένης επίλυσης, κοινώς να επιταχύνουν αναζητήσεις που αναφέρονται και σε αρχεία που δεν υπάρχουν. Αυτό αποτελεί το λεγόμενο negative dentry.

5 Η Page cache του Linux

Γενικά οι προσβάσεις για ανάγνωση/εγγραφή δεδομένων από συσκευές μόνιμης αποθήκευσης είναι πολύ πιο αργές από τις αντίστοιχες προσβάσεις στην κεντρική μνήμη RAM του υπολογιστή. Για αυτό το λόγο το Linux, όπως και όλα τα σύγχρονα ΛΣ, κάνει ότι μπορεί για να αποφεύγει όσο το δυνατό τις προσβάσεις αυτές. Για αυτό το σκοπό υπάρχει η page cache η οποία χρησιμοποιείται για την προσωρινή αποθήκευση δεδομένων τα οποία βρίσκονται σε κάποια μονάδα μόνιμης αποθήκευσης.

Στην κατασκευή μιας τέτοιας προσωρινής δομής αποθήκευσης, υπάρχουν ορισμένες παράμετροι που χρειάζεται να λάβουμε υπόψη, τις οποίες εξετάζουμε στις επόμενες δύο υποενότητες.

Μεταφορά σελίδων προς την page cache

Αναφέρουμε πρώτα τα βασικά χαρακτηριστικά των συσκευών αποθήκευσης:

- Bandwidth: εύρος ζώνης της συσκευής, δηλαδή ο όγκος δεδομένων που μπορεί να μεταφερθεί στην μονάδα χρόνου.
- Latency: Χρόνος από το αίτημα της πρόσβασης μέχρι την ικανοποίηση του.
- Κόστος ανά μονάδα αποθηκευμένης πληροφορίας: Εξαρτάται από την τεχνολογία του μέσου αποθήκευσης.
- Πυκνότητα: Η χωρητικότητα της συσκευής προς το (φυσικό) μέγεθος της ή το εμβαδό των ολοκληρωμένων κυκλωμάτων της.

Η DRAM προσφέρει μεγαλύτερο bandwidth και μικρότερο latency σε σχέση με μη πτητικά μέσα αποθήκευσης, και για αυτό λέμε άτυπα ότι οι προσβάσεις στην DRAM είναι πιο γρήγορες. Παράλληλα όμως, η DRAM διαθέτει μεγαλύτερο κόστος και μικρότερη πυκνότητα. Λόγω αυτού, ιδανικά θέλουμε να έχουμε όσες περισσότερες χρησιμοποιούμενες σελίδες γίνεται στην page cache, αλλά πρακτικά είμαστε περιορισμένοι από την διαφορά μεγέθους στην χωρητικότητα μεταξύ RAM και δίσκου.

Από τα παραπάνω, γίνεται αντιληπτή η ανάγκη για προσδιορισμό μιας πολιτικής αντικατάστασης σελίδων όταν η page cache λειτουργεί στο όριο της χωρητικότητας της. Ευτυχώς για τους συντηρητές των συστημάτων αρχείων, θεωρούμε ότι αυτή η υποδομή είναι έτοιμη από το υποσύστημα διαχείρισης μνήμης του Linux, και η διαχείριση σελίδων στο επίπεδο της page cache γίνεται με διαφανή τρόπο κατά την ανάκτηση των σελίδων που χρειαζόμαστε.

Κάποιες εφαρμογές, με κοινό παράδειγμα να αποτελούν τα συστήματα βάσεων δεδομένων, δεν αρκούνται στην πολιτική που έχει υλοποιήσει το υποσύστημα memory management του Linux, και για αυτό επιλέγουν να υλοποιήσουν δική τους page cache (για παράδειγμα εδώ), προσπερνώντας την ήδη υπάρχουσα page cache μέσω της σημαίας *O_DIRECT*.

Ενημέρωση σελίδων από την page cache

Είναι πολύ πιθανό μια σελίδα που φέρνουμε στην page cache να δεχτεί τροποποίηση, περίπτωση στην οποία πλέον η page cache διαθέτει ένα ενημερωμένο αντίγραφο, που είναι αναγκαίο κάποια στιγμή να λάβει και το μέσο αποθήκευσης. Σημειώνουμε μια τέτοια σελίδα ως dirty.

Το επιθυμητό θα ήταν να καθυστερούμε κατά το μέγιστο δυνατό το συγχρονισμό του μέσου αποθήκευσης με την page cache, ώστε να ομαδοποιούμε / συγχωνεύουμε

προσβάσεις στο επίπεδο της page cahce, μειώνοντας έτσι τις προσβάσεις στο μέσο. Έχουμε όμως δύο κύριους παράγοντες που μας πιέζουν να ενημερώνουμε τακτικά το μέσο αποθήκευσης:

- Η page cache μπορεί να δουλεύει στο όριο της χωρητικότητας της, οπότε είμαστε αναγκασμένοι να διώχνουμε σελίδες που πιθανώς να είναι dirty.
- Πάντα υπάρχει ο κίνδυνος διακοπής τροφοδοσίας, οπότε, προετοιμασμένοι για αυτό το σενάριο, επιλέγουμε να μην κρατάμε για περισσότερο από ένα συγκεκριμένο διάστημα μαρκαρισμένες σελίδες στην page cache.

Αν ο χρήστης θέλει να έχει έλεγχο ως προς τον συγχρονισμό του μέσου αποθήκευσης, μπορεί να αξιοποιήσει τις ειδικές για αυτό τον σκοπό κλήσεις συστήματος, ονομαστικά τις `sync`, `fsync`, και `fdatasync`.

Παραδείγματα διαμεσολάβησης της page cache στο σύστημα αρχείων *ext2*

Για όποιον θέλει να έχει μια εικόνα αλληλουχίας κλήσεων στο ext2, και πως αυτές οδηγούν στον κώδικα που αφορά την page cache, παρουσιάζουμε τις επόμενες δύο περιπτώσεις:

Ανάγνωση αρχείου

`ext2_file_read_iter` → `generic_file_read_iter` → `filemap_read` → `filemap_get_pages` → `filemap_get_read_batch`

Ανάγνωση directory

`ext2_readdir` → `ext2_get_folio` → `read_mapping_folio` → `read_cache_folio`

Γενικά, μεταξύ και άλλων ειδών σελίδων, στην page cache μπορούμε να έχουμε σελίδες αρχείων, σελίδες καταλόγων, αλλά και σελίδες στις οποίες αποθηκεύουμε δεδομένα διεργασιών ώστε να μεταφερθούν στον δίσκο υπό συνθήκες πίεσης του συστήματος (στο λεγόμενο swap space). Το τελευταίο παράδειγμα βέβαια είναι ξένο ως προς τα συστήματα αρχείων.

6 Το σύστημα αρχείων *ext2-lite*

Στα πλαίσια του δεύτερου μέρους της εργαστηριακής άσκησης σας δίνουμε κώδικα ο οποίος προέρχεται απο το κομμάτι του πυρήνα του Linux που υλοποιεί το σύστημα αρχείων ext2. Έχουμε αφαιρέσει μεγάλα κομμάτια κώδικα που αφορούσαν χαρακτηριστικά με τα οποία δε θα ασχοληθούμε στην άσκηση. Σκοπός της άσκησης είναι να επικεντρωθείτε και να εξοικειωθείτε με τη διεπαφή του VFS και να κατανοήσετε πώς μία υλοποίηση ενός συγκεκριμένου συστήματος αρχείων μπορεί να ενταχθεί στο Linux και πως πρέπει να διαχειρίζεται τις διάφορες δομές του VFS με σωστό τρόπο. Θα ονομάσουμε το "νέο" αυτό σύστημα αρχείων που θα υλοποιήσετε `ext2-lite` καθώς είναι ένα μικρό υποσύνολο του ext2. Φυσικά όποιος επιθυμεί να δει πως ακριβώς υλοποιείται το πλήρες ext2 μπορεί να δει τον κώδικα που υπάρχει στον φάκελο `fs/ext2` του πυρήνα του Linux. Εδώ θεωρούμε πως έχετε ήδη ασχοληθεί με το πρώτο κομμάτι της εργαστηριακής άσκησης οπότε έχετε κατανοήσει αρκετά το πως οργανώνει τα δεδομένα σε ένα δίσκο το ext2 και σας είναι γνωστά πράγματα όπως τι είναι το superblock του ext2 και τι πληροφορίες περιέχει, τι είναι ένα block group, τι είναι το inode bitmap, κλπ.

6.1 Το module *ext2-lite*

Ο κώδικας που σας δίνεται περιλαμβάνει ένα kernel module το οποίο μπορείτε να μεταγλωτίσετε και να εισάγετε στον πυρήνα με τον τρόπο που το κάνατε και στην προηγούμενη εργαστηριακή άσκηση. Αφού έχετε κατεβάσει τον βοηθητικό κώδικα απο το helios μπορείτε να τρέξετε:

```
$ tar xvfz ext2-lite-helpcode.tgz
$ cd ext2-lite-helpcode
$ make
# insmod ./ext2-lite
```

Οι συναρτήσεις που εκτελούνται όταν εισάγουμε και εξάγουμε το module απο τον πυρήνα είναι οι `init_ext2_fs` και `exit_ext2_fs`. Μέσα στην `init_ext2_fs` γίνεται η καταχώρηση (registration) του `ext2-lite` στον πυρήνα και αντίστοιχα στην `exit_ext2_fs` γίνεται η διαγραφή του. Αφού έχετε εισάγει το module στον πυρήνα και αφού έχετε συμπληρώσει σωστά αυτές τις δύο συναρτήσεις θα πρέπει να βλέπετε στο `/proc/filesystems` το νέο υποστηριζόμενο σύστημα αρχείων:

```
$ cat /proc/filesystems
...
        ext2-lite
...
```

6.2 Οι βασικές δομές του *ext2-lite*

Πριν αρχίσουμε να βλέπουμε τον κώδικα του `ext2-lite` είναι βασικό να κατανοήσουμε τις δομές που χρησιμοποιεί για την αποθήκευση των βασικών δομών όπως είναι το superblock, το inode, τα block group descriptors, κλπ. Όπως και σε πολλά άλλα κομμάτια του πυρήνα του Linux έτσι και στο κομμάτι των συστημάτων αρχείων θα δούμε για κάποια δεδομένα υπάρχουν αντίγραφα σε παραπάνω απο μία θέσεις μνήμης. Για παράδειγμα, θα δούμε οτι για το superblock του ext2 ο πυρήνας διατηρεί μία δομή στην μνήμη του (τη δομή `ext2_sb_info`) αλλά επίσης έχει στην page cache του και έναν buffer που αντιστοιχίζεται στο superblock του συστήματος αρχείων στο δίσκο. Το ίδιο ισχύει και για τη δομή inode του ext2. Οι βασικές δομές που χρησιμοποιούνται στο `ext2-lite` και ορίζονται στο αρχείο `ext2.h` είναι:

- `ext2_sb_info`: Η δομή στη μνήμη του πυρήνα η οποία διατηρεί όποια πληροφορία είναι απαραίτητη για ένα προσαρτημένο σύστημα αρχείων `ext2-lite`. Αρχικοποιείται κατά την προσάρτηση του συστήματος αρχείων και παραμένει στη μνήμη του πυρήνα μέχρι να γίνει αποπροσάρτηση.

- `ext2_super_block`: Η δομή η οποία αναπαριστά επακριβώς ένα ext2 superblock όπως αυτό αποθηκεύεται στο δίσκο. Τα πεδία αυτής της δομής είναι ίδια ένα προς ένα με τα πεδία που ορίζονται απο το ext2 για το τι περιέχει ένα superblock. Στην ουσία για να διαβάσουμε το ext2 superblock απο τον δίσκο στην page cache του πυρήνα χρησιμοποιούμε ένα `buffer_head` το οποίο αποθηκεύεται στο πεδίο

  `s_sbh` της δομής `ext2_sb_info`. Για εύκολη άμεση πρόσβαση σε κάθε πεδίο του superblock χρησιμοποιείται το πεδίο `s_es` της ίδιας δομής το οποίο είναι ένας δείκτης που δείχνει στα περιεχόμενα του `s_sbh buffer_head`.

- `ext2_inode_info`: Αντίστοιχα με τις δομές για το superblock, για τα ext2 inodes έχουμε μία δομή η οποία υπάρχει μόνο στη μνήμη του πυρήνα και μία η οποία υπάρχει στην page cache του πυρήνα αλλα αναπαριστά επακριβώς ένα ext2 inode στο δίσκο. Η δομή που υπάρχει στη μνήμη του πυρήνα είναι η `ext2_inode_info`.

- `ext2_inode`: Η δομή που αναπαριστά επακριβώς ένα ext2 inode όπως αυτό αποθηκεύεται στο δίσκο. Τα πεδία του είναι ίδια ένα προς ένα με τα πεδία που ορίζονται απο το ext2 για το τι περιέχει ένα inode. Για το διάβασμα ενός inode απο τον δίσκο χρησιμοποιείται η συνάρτηση `sb_bread` για το διάβασμα του block στο οποίο περιέχεται το συγκεκριμένο inode. Η `sb_bread` επιστρέφει ένα `buffer_head` τα δεδομένα του οποίου περιέχουν τα δεδομένα του συγκεκριμένου block. Οπότε χρησιμοποιούμε έναν δείκτη σε `ext2_inode` ώστε να δείξουμε στο κατάλληλο σημείο μέσα στα δεδομένα αυτά.

- `ext2_dirent`: Η δομή που χρησιμοποιείται για να δεικτοδοτήσει στη μνήμη έναν buffer που περιέχει τα δεδομένα ενός directory entry που βρίσκεται στον δίσκο ενός ext2 συστήματος αρχείων. Χρησιμοποιείται για την αναζήτηση ονομάτων αρχείων μέσα στους καταλόγους του ext2.

- `ext2_group_desc`: Η δομή που αναπαριστά επακριβώς ένα ext2 group descriptor. Τα πεδία του είναι ίδια ένα προς ένα με τα πεδία που ορίζονται απο το ext2 για έναν group descriptor. Η δομή αυτή χρησιμοποιείται για την δεικτοδότηση buffers που έχουν έρθει στην page cache του πυρήνα απο blocks του δίσκου που περιέχουν τα group descriptors.

6.3 Προσάρτηση/Αποπροσάρτηση ενός *ext2-lite* συστήματος αρχείων

Πλέον μπορείτε να προσαρτήσετε ένα σύστημα αρχείων `ext2-lite` στο σύστημά σας. Με τις παρακάτω εντολές δημιουργούμε ένα σύστημα αρχείων ext2 (αφαιρώντας κάποια επιπλέον features με το flag `-O none` ώστε να μπορεί να προσαρτηθεί και ως ext2-lite) και το προσαρτούμε στον φάκελο `mnt`. Με τη χρήση του flag `-m 0` απενεργοποιείται η χρήση reserved blocks για κάποιον συγκεκριμένο χρήστη (συνήθως τον root).

```
$ touch ext2-lite.img && truncate -s 128M ext2-lite.img
$ mkfs.ext2 -b 1024 -L "ext2-lite fs" -O none -m 0 ./ext2-lite.img
$ mount -t ext2-lite -o loop ./ext2-lite.img /mnt
```

Κατα τη στιγμή της προσάρτησης του συστήματος αρχείων, εκτελείται η συνάρτηση που έχει δηλωθεί στο πεδίο `.mount` της δομής `file_system_type` που δώσαμε στον πυρήνα κατα την καταχώρηση του συστήματος αρχείων μας, στη συγκεκριμένη περίπτωση η `ext2_mount`. Συγκεκριμένα για συστήματα αρχείων που βρίσκονται σε συσκευές block ο πυρήνας προσφέρει τη γενική συνάρτηση `mount_bdev` η οποία αναλαμβάνει την περισσότερη απο τη δουλεία που πρέπει να γίνει και αφήνει στο εκάστοτε σύστημα αρχείων μόνο το κομμάτι που αφορά την αρχικοποίηση των διαφόρων πεδίων του superblock. Για αυτό δέχεται σαν τελευταίο όρισμα έναν δείκτη σε μία συνάρτηση `fill_super`. Ο ρόλος αυτής της συνάρτησης είναι να αρχικοποιήσει κατάλληλα το παρεχόμενο superblock. Στην περίπτωση του `ext2-lite` η συνάρτηση αυτήν είναι η `ext2_fill_super`. Τα βασικά βήματα που κάνει η `ext2_fill_super` είναι:

- Δέσμευση μνήμης για μία δομή `ext2_sb_info`, η οποία στην ουσία είναι ένα αντίγραφο στη μνήμη του πυρήνα του superblock που βρίσκεται στον δίσκο.

- Ορισμός του κατάλληλου blocksize.

- Εντοπισμός του block στο δίσκο όπου αποθηκεύται το superblock.
- Διάβασμα του superblock απο το δίσκο. Αυτό γίνεται με τη συνάρτηση `sb_bread` η οποία επιστρέφει το `buffer_head` που διαβάστηκε.
- Διάβασμα των blocks που περιέχουν τους group descriptors απο τον δίσκο. Κι εδώ με τη χρήση της `sb_bread`.
- Καταχώρηση της δομής `ext2_sops` στο πεδίο `s_op` του VFS superblock. Έτσι ο πυρήνας θα μπορεί να καλέσει τις αντίστοιχες συναρτήσεις του `ext2-lite` όποτε χρειάζεται.
- Εντοπισμός του root inode, δημιουργία ενός dentry για το root inode και αποθήκευσή του στο πεδίο `s_root` του VFS superblock. Το πεδίο αυτό χρησιμοποιείται απο το VFS κατα την αναζήτηση ενός pathname που περιλαμβάνει κάποιο mountpoint με το σύστημα αρχείων μας.

Αν όλα πάνε καλά και η `ext2_mount` επιστρέψει χωρίς κάποιο σφάλμα, το σύστημα αρχείων μας έχει προσαρτηθεί με επιτυχία και μπορούμε να το δούμε στο `/proc/mounts`.

```
$ cat /proc/mounts
...
/dev/loop0 /mnt ext2-lite rw,relatime,errors=continue 0 0
...
```

6.4 Η δομή *ext2_sops*

Τώρα που το σύστημα αρχείων μας έχει προσαρτηθεί στο σύστημα, το VFS superblock που αρχικοποιήσαμε στην `ext2_fill_super` έχει προστεθεί στον πυρήνα ο οποίος πλέον θα καλεί τις συναρτήσεις που έχουμε ορίσει στη δομή `ext2_sops` όποτε χρειάζεται. Συγκεκριμένα η δομή `ext2_sops` ορίζει τα παρακάτω πεδία:

- `.alloc_inode = ext2_alloc_inode`

  Όπως αναφέρθηκε και νωρίτερα η συγκεκριμένη μέθοδος καλείται όταν ο πυρήνας χρειάζεται να δεσμεύσει μνήμη για ένα inode. Συγκεκριμένα η `ext2_alloc_inode` δεσμεύει μνήμη για ένα `ext2_inode_info`, το οποίο περιέχει μέσα του ένα VFS inode.

- `.free_inode = ext2_free_inode_sb`

  Καλείται για την αποδέσμευση της μνήμης ενός inode. Συγκεκριμένα η `ext2_free_inode_sb` αποδεσμεύει τη μνήμη που καταλαμβάνει ένα `ext2_inode_info` το οποίο περιέχει μέσα του το VFS inode.

- `.write_inode = ext2_write_inode`

  Καλείται όταν κάποιο VFS inode που βρίσκεται στη μνήμη πρέπει να γραφτεί στον δίσκο. Συγκεκριμένα η `ext2_write_inode` καλεί την `ext2_get_inode` ώστε να εντοπίσει το inode στο δίσκο που συνδέεται με αυτό στη μνήμη και συνέχεια αντιγράφει τα δεδομένα αυτά στο inode που αποθηκεύεται στο δίσκο.

- `.evict_inode = ext2_evict_inode`

  Καλείται απο το VFS όταν για κάποιο inode ο αριθμός των links προς αυτό έχει μηδενιστεί. Ευθύνη της συνάρτησης είναι να απελευθερώσει τα blocks του δίσκου που χρησιμοποιούνται απο το συγκεκριμένο inode.

- `.put_super = ext2_put_super`

  Καλείται όταν το VFS θέλει να αποδεσμεύσει τη μνήμη που καταλαμβάνεται απο κάποιο superblock, δηλαδή κατα τη διαδικασία της αποπροσάρτησης (umount). Συγκεκριμένα η `ext2_put_super` αρχικά εντοπίζει τον αντίστοιχο buffer στη μνήμη που συνδέεται με το superblock στο δίσκο μέσω του πεδίου `s_es` της δομής `ext2_sb_info`, ανανεώνει τα κατάλληλα πεδία (αν το σύστημα αρχείων δεν έχει προσαρτηθεί μόνο για ανάγνωση), μαρκάρει τον buffer ως dirty (με την `mark_buffer_dirty`) και αν απαιτείται περιμένει μέχρι να εγγραφεί ο buffer πίσω στον δίσκο (με την `sync_dirty_buffer`). Στη συνέχεια, αποδεσμεύει τη μνήμη όλων των buffers που χρησιμοποιούσε και τη μνήμη όσων πεδίων το απαιτούν. Τέλος αποδεσμεύεται η μνήμη της δομής `ext2_sb_info`. Την αποδέσμευση του VFS superblock αναλαμβάνει το VFS.

- `.sync_fs = ext2_sync_fs`

  Καλείται όταν το VFS θέλει να μεταφέρει στο δίσκο τα δεδομένα του superblock που είναι σε κατάσταση dirty στη μνήμη. Συγκεκριμένα η `ext2_sync_fs` ελέγχει αν έχει γίνει 1 το bit `EXT2_VALID_FS` στο πεδίο `s_state` της δομής `ext2_super_block` και αν ναι το μηδενίζει (γιατί θέλουμε όσο ένα ext2-lite σύστημα αρχείων είναι προσαρτημένο αυτό να είναι 0). Στη συνέχεια ανανεώνει τα κατάλληλα πεδία (αν το σύστημα αρχείων δεν έχει προσαρτηθεί μόνο για ανάγνωση), μαρκάρει τον buffer ως dirty (με την `mark_buffer_dirty`) και αν απαιτείται περιμένει μέχρι να εγγραφεί ο buffer πίσω στον δίσκο (με την `sync_dirty_buffer`).

- `.statfs = ext2_statfs`

  Καλείται απο το VFS για να πάρει στατιστικά για το σύστημα αρχείων. Συγκεκριμένα η `ext2_statfs` βρίσκει το αντίστοιχο ext2 superblock μέσω της παραμέτρου `dentry` και ενημερώνει κατάλληλα τα πεδία της δομής `kstatfs` που δίνεται ως δεύτερη παράμετρος στη συνάρτηση.

- `.remount_fs = ext2_remount`

  Καλείται απο το VFS όταν χρειάζεται σε κάποιο ήδη προσαρτημένο σύστημα αρχείων να αλλάξουν κάποιες απο τις παραμέτρους προσάρτησης.

- `.show_options = ext2_show_options`

  Καλείται απο το VFS για να διαβάσει τις παραμέτρους με τις οποίες έχει προσαρτηθεί ένα σύστημα αρχείων.

6.5 Πρόσβαση στα ext2 inodes απο το VFS

Αφού προσαρτήσουμε ένα σύστημα αρχείων `ext2-lite`, πλέον ο πυρήνας έχει πρόσβαση στο root dentry του (είναι αυτό που έχει επιστραφεί απο την `ext2_mount`). Έτσι, όταν πλέον σε κάποιο pathname εμφανιστεί ένα ext2-lite mountpoint ο πυρήνας μπορεί να βρει το root inode του συστήματος αρχείων μας και το οποίο περιέχει και το αντίστοιχο VFS inode μέσα του (πεδίο `vfs_inode`). Το root inode του ext2-lite έχει αρχικοποιηθεί μέσα απο την `ext2_fill_super` με χρήση της συνάρτησης `ext2_iget` η οποία διαβάζει ένα ext2 inode απο τον δίσκο και αποθηκεύει τα δεδομένα του σε μία δομή inode στη μνήμη.

Συγκεκριμένα τα βήματα που κάνει η `ext2_iget` είναι:

- Δέσμευση μνήμης για ένα `struct inode` μέσω της συνάρτησης `iget_locked` που παρέχεται απο τον πυρήνα. Η `iget_locked` για τη δέσμευση της μνήμης του inode καλεί την `.alloc_inode` του `ext2_sops` οπότε το `struct inode` που επιστρέφεται ξέρουμε οτι περιέχεται μέσα σε ένα `struct ext2_inode_info` για αυτό και μπορούμε να κάνουμε τη μετατροπή χρησιμοποιώντας τη συνάρτηση `EXT2_I`. Η `EXT2_I` χρησιμοποιεί την συνάρτηση `container_of` του πυρήνα η οποία δεδομένου ενός δείκτη σε μία δομή επιστρέφει έναν δείκτη σε μία μεγαλύτερη δομή η οποία περιέχει εντός της την πρώτη. Στη συγκεκριμένη περίπτωση η δομή `inode` του VFS περιέχεται στην `ext2_inode_info` στο πεδίο `vfs_inode`.

- Διάβασμα του ext2 inode απο τον δίσκο με χρήση της `ext2_get_inode`. Η `ext2_get_inode` αρχικά υπολογίζει σε ποιο block group βρίσκεται το ζητούμενο inode και διαβάζει απο το δίσκο τον αντίστοιχο block group descriptor. Μέσα σε αυτόν βρίσκει το block όπου ξεκινάει το inode table του block group και με βάση αυτό υπολογίζει το block στο οποίο βρίσκεται το ζητούμενο inode. Αφού φέρει αυτό το block στην page cache επιστρέφει έναν δείκτη στο κατάλληλο σημείο των δεδομένων όπου ξεκινάνε τα δεδομένα του ζητούμενου inode.

- Αρχικοποίηση των πεδίων του `struct inode` του VFS. Σημαντικό σημείο εδώ είναι η αρχικοποίηση των πεδίων `inode->i_op`, `inode->i_fop` και `inode->i_mapping->a_ops` ανάλογα με τον τύπου του αρχείου που αφορά το inode. Παρακάτω στον οδηγό αναλύουμε τα συγκεκριμένα πεδία για κάθε τύπο αρχείου.

- Αρχικοποίηση των πεδίων του `struct ext2_inode_info`.

- Απελευθέρωση (μείωση του reference count τους κατα ένα) του `buffer_head` που χρησιμοποιήθηκε για το διάβασμα του inode απο τον δίσκο και του VFS inode και επιστροφή της συνάρτησης.

Το πεδίο `inode->i_op` του VFS inode ορίζει τις συναρτήσεις/μεθόδους για το inode και εξαρτάται απο το είδος του αρχείου που αφορά το inode.

Για inodes που αφορούν φακέλους (directories) το πεδίο `inode->i_op` δείχνει στη δομή `ext2_dir_inode_operations`:

- `.create = ext2_create`

  Αρχικά καλεί την `ext2_new_inode` για να εντοπίσει ένα ελεύθερο inode στο δίσκο και να δεσμεύσει μνήμη για τα αντίστοιχα `struct inode` και `struct ext2_inode_info`. Παρακάτω στον οδηγό εξηγούμε πως γίνεται ο εντοπισμός και η ανάθεση ενός ελεύθερου inode στο δίσκο. Στη συνέχεια θέτει κατάλληλα τα πεδία `i_op`, `i_fop` και `i_mapping->a_ops` του VFS inode, τον μαρκάρει ως dirty και τέλος καλεί την `ext2_add_nondir` για να εισάγει το directory entry του νέου αρχείου στον κατάλογο.

- `.lookup = ext2_lookup`

  Αναζητά μέσα στα directory entries του φακέλου το συγκεκριμένο dentry και επιστρέφει έναν δείκτη σε αυτό. Αρχικά ελέγχει αν το όνομα του dentry είναι εντός του επιτρεπόμενου μήκους του `ext2`. Στη συνέχεια καλεί την `ext2_inode_by_name` για να εντοπίσει (αν υπάρχει) το inode που αντιστοιχεί στο συγκεκριμένο όνομα εντός του φακέλου. Αν το συγκεκριμένο όνομα δε βρεθεί το αντίστοιχο inode τίθεται σε `NULL` ενώ αν βρεθεί καλεί την `ext2_iget` για να διαβάσει απο το δίσκο το αντίστοιχο ext2 inode. Τέλος, καλεί την `d_splice_alias` για να εισάγει στην dentry cache το νέο dentry.

- `.link = ext2_link`

  Δημιουργεί μία νέα σύνδεση ονόματος με κάποιο inode. Αρχικά βρίσκει το inode μέσα απο το ήδη υπάρχον dentry με χρήση της `d_inode`. Ανανεώνει το χρόνο δημιουργίας του inode και αυξάνει τον αριθμό των links του κατα ένα.

Τέλος προσθέτει το νέο όνομα στα directory entries του καταλόγου (καλώντας την `ext2_add_link`) και δημιουργεί το νέο VFS dentry καλώντας την `d_instantiate`.

- `.unlink = ext2_unlink`

  Διαγράφει μία υπάρχουσα σύνδεση ονόματος με κάποιο inode. Αρχικά βρίσκει το κατάλληλο dentry καλώντας την `ext2_find_entry` και στη συνέχεια το διαγράφει απο τα directory entries του καταλόγου καλώντας την `ext2_delete_entry`.

- `.symlink = ext2_symlink`

  Δημιουργεί ένα symbolic link εντός του καταλόγου. Αρχικά ελέγχει οτι το μονοπάτι στο οποίο δείχνει το symbolic link χωράει σε ένα block του συστήματος αρχείων και αν όχι επιστρέφει σφάλμα. Αν ναι, καλεί την `ext2_new_inode` για να εντοπίσει και να δεσμέυσει ένα ελεύθερο ext2 inode απο τον δίσκο. Στη συνέχεια, αρχικοποιεί κατάλληλα τα πεδία του νέου inode διαχωρίζοντας ανάμεσα σε ”fast” και ”slow” symbolic links ανάλογα με το μήκος του μονοπατιού στο οποίο δείχνει ο σύνδεσμος. Στην πρώτη περίπτωση, το μονοπάτι αποθηκεύεται απευθείας μέσα στο inode στο πεδίο `i_link`. Αλλιώς το μονοπάτι αποθηκεύεται σε κάποιο data block του δίσκου το οποίο και σχετίζεται με το inode του συνδέσμου.

- `.mkdir = ext2_mkdir`

  Δημιουργεί μία νέα σύνδεση ονόματος η οποία αφορά έναν κατάλογο. Αρχικά αυξάνει τον αριθμό των links του καταλόγου στον οποίο θα δημιουργηθεί η νέα σύνδεση (αφήνουμε σαν ερώτημα το γιατί πρέπει να γίνει αυτό εδώ) και εντοπίζει και δεσμεύει ένα ελεύθερο ext2 inode στο δίσκο καλώντας την `ext2_new_inode`. Αφού αρχικοποιήσει κατάλληλα το νέο inode, αρχικοποιεί και τα αντίστοιχα data blocks του inode καλώντας την `ext2_make_empty`. Τέλος, προσθέτει τη νέα σύνδεση ονόματος στον γονικό κατάλογο καλώντας την `ext2_add_link` και αρχικοποιεί το VFS dentry καλώντας την `d_instantiate_new`.

- `.rmdir = ext2_rmdir`

  Διαγράφει τη σύνδεση ονόματος ενός (κενού) καταλόγου απο τον γονικό κατάλογο. Αρχικά ελέγχει οτι ο κατάλογος είναι όντως κενός (αγνοώντας τα `.` και `..`) και αν δεν είναι επιστρέφει κωδικό σφάλματος. Σε διαφορετική περίπτωση καλεί την `ext2_unlink` η οποία κάνει όλη τη δουλειά της διαγραφής της σύνδεσης ονόματος με τον ζητούμενο κατάλογο απο τον γονικό κατάλογο.

- `.mknod = ext2_mknod`

  Δημιουργεί μία νέα σύνδεση ονόματος η οποία αφορά ένα ειδικό αρχείο, όπως ένα αρχείο συσκευής χαρακτήρων, συσκευής block, named pipe κλπ. Παρόλο που έχουμε αφήσει στον κώδικα του `ext2-lite` ότι αφορά τη διαχείριση των ειδικών αρχείων για πληρότητα, δε θα ασχοληθούμε με αυτά στα πλαίσια της άσκησης.

- `.rename = ext2_rename`

  Αλλαγή μίας σύνδεσης ονόματος σε μία καινούρια για ένα συγκεκριμένο inode. Αρχικά εντοπίζει τη θέση του αρχικού `ext2_dirent` καλώντας την `ext2_find_entry`. Αν το αρχικό inode αφορά κατάλογο τότε βρίσκουμε τον γονικό του κατάλογο μέσω της `ext2_dotdot`. Στη συνέχεια έχουμε δύο σενάρια ανάλογα με τον αν το νέο inode υπάρχει ήδη ή όχι. Στην πρώτη περίπτωση εντοπίζεται το ήδη υπάρχον `ext2_dirent` μέσω της `ext2_find_entry` και ενημερώνεται κατάλληλα η σύνδεση με το inode του ώστε να δείχνει στο inode που μετονομάζουμε. Στην περίπτωση που το νέο inode δεν υπήρχε τότε δημιουργούμε απλά μία νέα σύνδεση ονόματος ανάμεσα στο νέο dentry και στο παλιό inode καλώντας την `ext2_add_link`.

- `.getattr = ext2_getattr`

  Καλείται απο το VFS για τη συλλογή των χαρακτηριστικών κάποιου inode, όπως για παράδειγμα σε ποιο χρήστη/γκρουπ ανήκει, το μέγεθός του κλπ. Το ext2 χρησιμοποιεί την `ext2_getattr` για όλους τους τύπους inode (κανονικό αρχείο, κατάλογος, ειδικό αρχείο, κλπ). Καλεί την γενική συνάρτηση του πυρήνα `generic_fillattr`. Δε θα ασχοληθούμε στα πλαίσια της άσκησης με αυτήν τη μέθοδο αλλά είναι απαραίτητη για να μπορούμε να βλέπουμε πληροφορίες για τα inodes των συστημάτων αρχείων μας.

- `.setattr = ext2_setattr`

  Καλείται απο το VFS για να τροποποιήσει χαρακτηριστικά κάποιου inode. Όπως και για την `.getattr` έτσι κι εδώ το ext2 χρησιμοποιεί την `ext2_setattr` για όλους τους τύπους inode. Δε θα ασχοληθούμε στα πλαίσια της άσκησης με αυτήν τη μέθοδο αλλά είναι απαραίτητη για να μπορούμε να τροποποιούμε χαρακτηριστικά των inodes των συστημάτων αρχείων μας.

Όσον αφορά inodes που αφορούν κανονικά αρχεία (regular files) και ειδικά αρχεία (αρχεία συσκευών χαρακτήρων, block, κλπ) το ext2 δεν χρειάζεται να κάνει κάτι παραπάνω απο όσα κάνει ήδη το VFS οπότε στις αντίστοιχες δομές `ext2_file_inode_operations` και `ext2_special_inode_operations` ορίζονται μόνο οι μέθοδοι `.getattr` και `.setattr`.

6.6 Διαχείριση inodes στο *ext2-lite*

Στο αρχείο `ialloc.c` πάρεχονται οι συναρτήσεις για τη διαχείριση των inodes στο σύστημα αρχείων `ext2-lite`. Οι βασικές συναρτήσεις είναι οι παρακάτω:

- `ext2_new_inode`

  Η συνάρτηση η οποία χρησιμοποιείται για τον εντοπισμό και τη δέσμευση ενός ελεύθερου inode απο τον δίσκο κατα τη δημιουργία ενός νέου αρχείου. Αρχικά σκανάρει τα inode bitmaps των block groups ξεκινώντας απο το block group στο οποίο βρίσκεται ο γονικός κατάλογος του νέου αρχείου και σταματάει μόλις βρει κάποιο διαθέσιμο inode. Στη συνέχεια ενημερώνει το αντίστοιχο inode bitmap θέτοντας το κατάλληλο bit. Τέλος, αρχικοποιεί κατάλληλα το αντίστοιχο `ext2_inode_info`, συμπεριλαμβανομένου του VFS inode που περιέχεται μέσα σε αυτό, και επιστρέφει έναν δείκτη σε αυτό το VFS inode.

- `ext2_free_inode`

  Η συνάρτηση η οποία χρησιμοποιείται για την απελευθέρωση ενός inode στον δίσκο κατα τη διαγραφή ενός αρχείου. Εδώ τα πράγματα είναι πιο απλά σε σύγκριση με την `ext2_new_inode` καθώς το μόνο που πρέπει να κάνουμε είναι να βρούμε το inode bitmap του block group στο οποίο ανήκει το συγκεκριμένο inode και να μηδενίσουμε το κατάλληλο bit.

- `ext2_count_free_inodes`

  Επιστρέφει τον συνολικό αριθμό διαθέσιμων inodes στο σύστημα αρχείων σας. Υπάρχουν δύο τρόποι να εξαχθεί αυτή η πληροφορία. Ο πρώτος είναι να διαβάσουμε απλά όλα τα block group descriptors και να αθροίσουμε όλα τα πεδία `bg_free_inodes_count`. Ο δεύτερος τρόπος είναι να διαβάσουμε όλα τα inode bitmaps και να αθροίσουμε όλα τα μη μηδενικά bits. Στο ext2-lite έχουμε κρατήσει μόνο τον πρώτο τρόπο για να απλοποιήσουμε όσο το δυνατόν τον κώδικα ωστόσο μπορείτε αν θέλετε να υλοποιήσετε και τον δεύτερο τρόπο ώστε να εξοικειωθείτε με τη διαχείριση bitmaps στον πυρήνα.

- `ext2_count_dirs`

  Επιστρέφει τον αριθμό των καταλόγων που υπάρχουν στο σύστημα αρχείων αθροίζοντας τα πεδία `bg_used_dirs_count` όλων των block group descriptors.

6.7 Διαχείριση blocks στο *ext2-lite*

Στο αρχείο `balloc.c` πάρεχονται οι συναρτήσεις για τη διαχείριση των blocks στο σύστημα αρχείων `ext2-lite`. Οι βασικές συναρτήσεις είναι οι παρακάτω:

- `ext2_new_blocks`

  Η συνάρτηση η οποία χρησιμοποιείται για τον εντοπισμό και τη δέσμευση ενός ελεύθερου block στο δίσκο. Έχουμε απλοποιήσει αρκετά τη διαδικασία εντοπισμού ελεύθερου block σε σχέση με το ext2. Πιο συγκεκριμένα στο ext2-lite σκανάρουμε απλά με τη σειρά τα block bitmaps, ξεκινώντας απο το block group στο οποίο ανήκει το inode, και εντοπίζουμε το πρώτο διαθέσιμο block. Αφού το εντοπίσουμε, το δεσμεύουμε θέτοντας το κατάλληλο bit στο block bitmap.

- `ext2_free_blocks`

  Η συνάρτηση η οποία χρησιμοποιείται για την απελευθέρωση ενός αριθμού απο (συνεχόμενα) blocks του δίσκου κατα τη διαγραφή των δεδομένων ενός inode. Αφού ελέγξει οτι τα blocks προς διαγραφή είναι έγκυρα (ελέγχοντας για παράδειγμα οτι δεν περιλαμβάνουν το superblock, κοκ) βρίσκει το αντίστοιχο block bitmap στο οποίο ανήκουν τα blocks και μηδενίζει τα κατάλληλα bits. Στο ext2 υπήρχε και έλεγχος για το ενδεχόμενο τα δεδομένα blocks να ανήκουν σε πολλαπλά block groups (group overflow) αλλά στο ext2-lite για να απλοποιήσουμε τον κώδικα δε λαμβάνουμε υπόψη αυτό το ενδεχόμενο.

- `ext2_get_group_desc`

  Επιστρέφει έναν δείκτη στη δομή `ext2_group_desc` που αναπαριστά ένα συγκεκριμένο block group. Αρχικά ελέγχει αν το ζητούμενο block group είναι έγκυρο, δηλαδή αν είναι μικρότερο απο το σύνολο των block groups. Στη συνέχεια υπολογίζει το offset του συγκεκριμένου block group descriptor μέσα στο block του δίσκου που το περιέχει και με βάση αυτό το offset επιστρέφει δείκτη στο κατάλληλο σημείο του buffer που αντιστοιχεί σε αυτό το block (το βρίσκει απο το πεδίο `s_group_desc` της δομής `ext2_sb_info`.

- `ext2_count_free_blocks`

  Επιστρέφει τον συνολικό αριθμό διαθέσιμων blocks στο σύστημα αρχείων σας. Υπάρχουν δύο τρόποι να εξαχθεί αυτή η πληροφορία. Ο πρώτος είναι να διαβάσουμε απλά όλα τα block group descriptors και να αθροίσουμε όλα τα πεδία `bg_free_blocks_count`. Ο δεύτερος τρόπος είναι να διαβάσουμε όλα τα block bitmaps και να αθροίσουμε όλα τα μη μηδενικά bits. Στο ext2-lite έχουμε κρατήσει μόνο τον πρώτο τρόπο για να απλοποιήσουμε όσο το δυνατόν τον κώδικα, ωστόσο μπορείτε αν θέλετε να υλοποιήσετε και τον δεύτερο τρόπο ώστε να εξοικειωθείτε με τη διαχείριση bitmaps στον πυρήνα.

6.8 Η διεπαφή *file_operations*

Όσον αφορά την προσπέλαση των δεδομένων των αρχείων ή/και των καταλόγων στο ext2 (και στο ext2-lite) τα πράγματα είναι αρκετά απλά. Συγκεκριμένα, το ext2 ορίζει τις δομές `ext2_file_operations` (για regular files) και `ext2_dir_operations` (για directories) οι οποίες περιέχουν δείκτες σε συναρτήσεις/μεθόδους οι οποίες καλούνται απο το VFS για να εξυπηρετήσουν κλήσεις συστήματος που αφορούν file descriptors (π.χ., `read`, `write`, `lseek`, ...). Και οι δύο αυτές δομές περιλαμβάνουν δείκτες σε γενικές συναρτήσεις του πυρήνα, π.χ., `generic_file_llseek`, `generic_file_read_iter`, κλπ (εκτός απο την `.iterate_shared` για directories η οποία υλοποιείται απο το ext2).

Αυτές οι γενικές συναρτήσεις του πυρήνα μπορούν να χρησιμοποιηθούν απο συστήματα αρχείων που χρησιμοποιούν συσκευές block υπο μία προυπόθεση: το σύστημα αρχείων πρέπει να παρέχει σε κάθε VFS inode που διαχειρίζεται ένα σύνολο συναρτήσεων/μεθόδων μέσω μίας δομής τύπου `struct address_space_operations`. Το σύνολο αυτών των μεθόδων αποθηκεύεται στο πεδίο `i_mapping->a_ops` των VFS inodes. Συγκεκριμένα, το ext2 ορίζει τη δομή `ext2_aops` (αρχείο `inode.c`). Αλλά τα πράγματα γίνονται ακόμα πιο απλά καθώς κάθε μία απο αυτές τις συναρτήσεις του ext2 τελικά καταλήγει να καλέσει μια αντίστοιχη γενική συνάρτηση του πυρήνα που κάνει τη δουλειά που πρέπει. Το ext2 αρκεί να παρέχει στον πυρήνα μία και μόνο συνάρτηση η οποία είναι υπεύθυνη για τον εντοπισμό ενός block μέσα στον δίσκο και την τοποθέτηση του σε έναν page cache buffer ώστε να μπορεί μετά το VFS να το προσπελάσει. Η συνάρτηση που κάνει αυτή τη δουλεία στον κώδικα του ext2 είναι η `ext2_get_block`.

7 Χρήσιμα Links

Linux Kernel VFS Documentation

Αναφορές

[1] Daniel Bovet and Marco Cesati, *Understanding the Linux Kernel, 3rd Edition*. O'Reilly Publications, 2005.