

Εργαστήριο Λειτουργικών Συστημάτων

Μάθημα 6^{ου} Εξαμήνου,

Τομέας Λογικού και Υπολογιστών

Ιωάννης Χατζηγιαννάκης

Πέμπτη 13 Μαΐου 2010
Αίθουσα ΒΑ



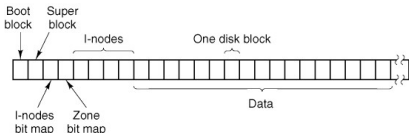
Σύστημα Αρχείων στο Λ.Σ. MINIX 3

- ▶ Όπως όλα τα Λ.Σ. και το MINIX 3 προσφέρει ένα σύστημα αρχείων για την αποθήκευση πληροφοριών
- ▶ Μπορεί να δεσμεύει / αποδεσμεύει αποθηκευτικό χώρο για τα αρχεία
- ▶ Να διαχειρίζεται τα blocks του δίσκου και να απελευθερώνει αποθηκευτικό χώρο
- ▶ Να διασφαλίζει την ασφάλεια των δεδομένων
- ▶ Πρόκειται για ένα 'μεγάλο' πρόγραμμα γραμμένο σε C – τρέχει εξ' ολοκλήρου στο user space
- ▶ Οι διεργασίες που θέλουν να διαβάσουν/γράψουν αρχεία στέλνουν μηνύματα στο σύστημα αρχείων (file system)
 - ▶ Το σύστημα αρχείων επεξεργάζεται το μήνυμα, εκτελεί τις απαραίτητες ενέργειες και επιστρέφει την απάντηση



Δομή Συστήματος Αρχείων

- ▶ Το Σύστημα Αρχείων του MINIX 3 αποτελείται από i-nodes, φακέλους και blocks δεδομένων
- ▶ Το μέγεθος των τμημάτων διαφοροποιείτε ανάλογα με το συνολικό μέγεθος του συστήματος και την υφιστάμενη τεχνολογία
- ▶ Όμως αυτά τα τμήματα βρίσκονται σε όλα τα συστήματα αρχείων τύπου MINIX 3



Το superblock του MINIX 3

- ▶ Το superblock περιέχει πληροφορίες για τη δομή του συστήματος αρχείων
- ▶ Έχει πάντα μέγεθος 1024 bytes
- ▶ Βάση του πλήθους των inodes και το μέγεθος των blocks μπορούμε να υπολογίσουμε το μέγεθος του bitmap των inodes
- ▶ Ο αποθηκευτικός χώρος διαιρείται σε zones από $\log_2 n$ blocks

Present on disk and in memory

Present in memory but not on disk

Number of i-nodes
(unused)
Number of i-node bitmap blocks
Number of zone bitmap blocks
First data zone
Log ₂ (block/zone)
Packing
Maximum file size
Number of zones
Magic number
padding
Block size (bytes)
FS sub-version
Pointer to i-node for root of mounted file system
Pointer to i-node mounted upon
i-nodes/block
Device number
Read-only flag
Native or byte-swapped flag
FS version
Direct zones/i-node
Indirect zones/indirect block
First free bit in i-node bitmap
First free bit in zone bitmap



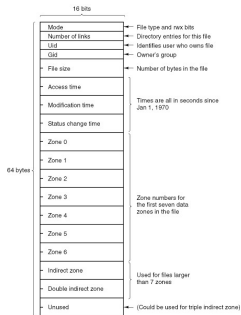
Bitmaps -- Καταγραφή Ελεύθερων Blocks

- ▶ Η καταγραφή του ελεύθερου χώρου γίνεται με την χρήση 2 bitmaps
- ▶ Όταν ένα αρχείο διαγράφεται ενημερώνεται το bitmap θέτοντας το bit που αντιστοιχεί στο block σε 0
- ▶ Όταν όλα τα blocks μιας ζώνης είναι ελεύθερα, με τον ίδιο τρόπο απελευθερώνεται και η ζώνη
- ▶ Για την δημιουργία ενός αρχείου, ανατρέχουμε στο bitmap για να εντοπίσουμε κενό χώρο και να τον δεσμεύσουμε
- ▶ Σχετικά με το i-node του καινούργιου αρχείου – για να επιταχυνθεί η διαδικασία, το superblock έχει ένα δείκτη στο πρώτο ελεύθερο i-node
 - ▶ Όταν το αρχείο διαγραφεί, το superblock ενημερώνεται για να δείχνει στο i-node του αρχείου

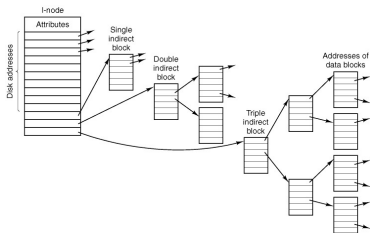


Το i-node του MINIX 3

- ▶ Τα i-nodes διατηρούν τις διευθύνσεις των blocks όπου αποθηκεύονται τα δεδομένα του αρχείου
- ▶ Αποθηκεύουν μεταδεδομένα που περιγράφουν το αρχείο
- ▶ Το μέγεθος ενός i-node είναι 64 bytes
- ▶ Η δομή επιτρέπει την αποθήκευση αρχείων έως 4 GB όταν το block size είναι 4 KB



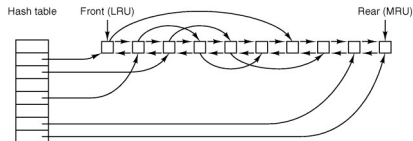
Απεικόνιση Αρχείων



- ▶ Τα triple indirect zone δεν υλοποιούνται στο MINIX 3

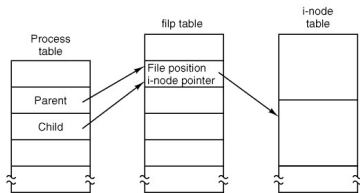


Δομή Block Cache στο MINIX 3



1. Πίνακας που περιέχει όλα τα hash values
 - ▶ Σε κάθε γραμμή αναθέτουμε μια συνδεδεμένη λίστα με τα blocks που έχουν την ίδια hash value
2. Συνδεδεμένη λίστα που περιέχει όλα τα blocks
 - ▶ Ταξινομημένη με βάση την παλαιότητα του block





- ▶ Σε ξεχωριστό 'διαμοιραζόμενο' πίνακα filp
- ▶ Ο πίνακας είναι κοινός για όλες τις διεργασίες

- ▶ Ο κώδικας του συστήματος αρχείου επικεντρώνεται στην εκτέλεση των κλήσεων του συστήματος
- ▶ Ας εξετάσουμε την περίπτωση όπου ένα πρόγραμμα εκτελεί: `n = read(fd, buffer, nbytes);`
- ▶ Μόλις παραλάβει το σύστημα αρχείων το μήνυμα
 - ▶ Ανατρέχει στον πίνακα filp σύμφωνα με την παράμετρο `fd`
 - ▶ Ελέγχει την τρέχουσα θέση στο αρχείο
 - ▶ Υπολογίζει το πλήθος των blocks που πρέπει να διαβαστούν – και τις διευθύνσεις τους
 - ▶ Για κάθε block ανατρέχει στην προσωρινή μνήμη ή μονάδα E/E
 - ▶ Συγχωνεύει τα blocks στον buffer
 - ▶ Αντιγράφει τον buffer στην μνήμη της διεργασίας
- ▶ Αφού ολοκληρωθεί η εκτέλεση της κλήσης, το σύστημα αρχείων διαβάζει ορισμένα block ακολουθώντας τεχνικές read-ahead

Σύνοψη 13⁷⁵ Διάλεξης

Προηγούμενο Μάθημα

Σύστημα Αρχείων

Δομή Συστήματος Αρχείων

Ανάγνωση Αρχείων

Λειτουργικό Σύστημα Minix

Διαχείριση Block

Διαχείριση i-Node

Διαχείριση Superblock, Πίνακας Ανοικτών Αρχείων, File Locks

Σύνοψη Μαθήματος

Σύνοψη Μαθήματος

Βιβλιογραφία

Επόμενη Διάλεξη

Διαχείριση Block

- ▶ Η δομή της προσωρινής μνήμης block ορίζεται στο αρχείο `/usr/src/servers/fs/buf.h`
- ▶ Ο πίνακας `buf` αποθηκεύει τα δεδομένα των block μαζί με όλες τις μετα-πληροφορίες
 - ▶ Μία εγγραφή χωρίζεται σε 2 τμήματα: `data` και `header`
 - ▶ Το τμήμα `data` αποτελείται από τα περιεχόμενα του block
 - ▶ Το τμήμα `header` περιέχει `pointers`, `counters`, `flags` που χαρακτηρίζουν την κατάσταση του block
- ▶ Το τμήμα δεδομένων ορίζεται ως `union` 7 διαφορετικών τύπων
 - ▶ Μερικές φορές βολεύει να προσπελάσουμε τα δεδομένα με την χρήση ειδικού τύπου πίνακα
 - ▶ π.χ., πίνακας χαρακτήρων, πίνακας `bit` ...
- ▶ Ο πίνακας με τις τιμές `hash` ονομάζεται `buf_hash`
- ▶ Οι συνδεδεμένες λίστες `LRU`, `MRU` τηρούνται από τους δείκτες `front` και `rear`

Εσωτερικές Δομές για Διαχείριση block (1)

```
EXTERN struct buf {
    /* Data portion of the buffer. */
    union { ... } b;

    /* Header portion of the buffer. */
    struct buf *b_next; /* link all free bufs in a chain */
    struct buf *b_prev; /* link all free bufs the other way */
    struct buf *b_hash; /* link bufs on hash chains */
    block_t b_blocknr; /* block number of its (minor)
    dev_t b_dev; /* major | minor device where block is
    char b_dir; /* CLEAN or DIRTY */
    char b_count; /* number of users of this buffer */
} buf[NR_BUFS];
```



Εσωτερικές Δομές για Διαχείριση block (2)

```
union {
    char b_data[_MAX_BLOCK_SIZE]; /* ordinary user
    /* directory block */
    struct direct b_dir[NR_DIR_ENTRIES(_MAX_BLOCK_SIZE)];
    /* V1 indirect block */
    zone1_t b_v1_ind[V1_INDIRECTS];
    /* V2 indirect block */
    zone2_t b_v2_ind[V2_INDIRECTS(_MAX_BLOCK_SIZE)];
    /* V1 inode block */
    d1_inode b_v1_ino[V1_INODES_PER_BLOCK];
    /* V2 inode block */
    d2_inode b_v2_ino[V2_INODES_PER_BLOCK(_MAX_BLOCK_SIZE)];
    /* bit map block */
    bch_t b_bitmap[FS_BITMAP_CHUNKS(_MAX_BLOCK_SIZE)];
} b;
```



Εσωτερικές Δομές για Διαχείριση block (3)

```
/* These defs make it possible to use
   bp->b_data instead of bp->b.b_data */
#define b_data      b.b_data
#define b_dir       b.b_dir
#define b_v1_ind    b.b_v1_ind
#define b_v2_ind    b.b_v2_ind
#define b_v1_ino    b.b_v1_ino
#define b_v2_ino    b.b_v2_ino
#define b_bitmap    b.b_bitmap
```



Εσωτερικές Δομές για Διαχείριση block (4)

```
/* the buffer hash table */
EXTERN struct buf *buf_hash[NR_BUF_HASH];

/* points to least recently used free block */
EXTERN struct buf *front;

/* points to most recently used free block */
EXTERN struct buf *rear;

/* # bufs currently in use (not on free list) */
EXTERN int bufs_in_use;
```



Λειτουργίες Διαχείρισης Block (1)

- ▶ Οι συναρτήσεις που διαχειρίζονται τις δομές της προσωρινής μνήμης block ορίζονται στο αρχείο `/usr/src/servers/fs/cache.h`
- ▶ 9 βασικές συναρτήσεις
- ▶ `get_block`: οποτεδήποτε κάποια συνάρτηση του συστήματος αρχείου χρειάζεται να διαβάσει ένα block
 - ▶ πρώτα ελέγχει την προσωρινή μνήμη: εξετάζει τον πίνακα με τις τιμές hash
 - ▶ αν δεν υπάρχει το block το μεταφέρει από την συσκευή E/E
 - ▶ πρέπει να επιλέξει ποιο block θα αφαιρέσει από τη cache (για να χωρέσει αυτό που μόλις μεταφέρθηκε)
- ▶ `put_block`: τοποθετεί ένα block στην προσωρινή μνήμη και σε ορισμένες περιπτώσεις το εγγράφει στην συσκευή E/E
 - ▶ ενημερώνει την λίστα LRU



Λειτουργίες Διαχείρισης Block (2)

- ▶ `alloc_zone`: Δέσμευση νέας ζώνης (για να μεγαλώσει ένα αρχείο)
 - ▶ αν ζητηθεί μόνο 1 block ελέγχει το πεδίο `s_zsearch` του `superblock`
 - ▶ αλλιώς ελέγχει το `zone bitmap` για την πρώτη ελεύθερη θέση
 - ▶ προσπαθεί να βρει μια ζώνη 'κοντά' στην τελευταία ζώνη που χρησιμοποιεί το αρχείο
 - ▶ αυτό γίνεται ξεκινώντας την αναζήτηση ανατρέχοντας στον πίνακα στην θέση της τελευταίας ζώνης
- ▶ `free_zone`: Αποδέσμευση ζώνης (κατά την διαγραφή αρχείου)
- ▶ `rw_block`: Μεταφορά block από/προς αποθηκευτική μονάδα από/προς cache
 - ▶ βασικότερη λειτουργία



Λειτουργίες Διαχείρισης Block (3)

- ▶ `invalidate`: Απομάκρυνση αποθηκευμένων block από προσωρινή μνήμη
 - ▶ χρησιμοποιείται όταν πρόκειται να κάνουμε update την συσκευή E/E
 - ▶ 'αδειάζει' την προσωρινή μνήμη
- ▶ `flushall`: Αποθήκευση όλων των block που έχουν αλλαγές
 - ▶ χρησιμοποιείται από την `get_block` όταν κάποιο block πρέπει να αφαιρεθεί από την προσωρινή μνήμη για να προκύψει χώρος για το νέο block
 - ▶ επίσης καλείται από την κλήση συστήματος SYNC – καλείται περιοδικά
- ▶ `rw_scattered`: Μεταφορά πολλαπλών block από/προς αποθηκευτική μονάδα και από/προς cache
- ▶ `rm_lru`: Διαγραφή του πρώτου block από την λίστα LRU



Λειτουργίες Διαχείρισης i-Node (1)

- ▶ Η δομή του i-node ορίζεται στο αρχείο `/usr/src/servers/fs/inode.h`
- ▶ Οι συναρτήσεις που διαχειρίζονται τα inodes ορίζονται στο αρχείο `/usr/src/servers/fs/inode.c`
- ▶ 10 βασικές συναρτήσεις
- ▶ Ορισμένες ακολουθούν την ίδια λογική με την διαχείριση block
- ▶ `get_inode`: οποτεδήποτε κάποια συνάρτηση του συστήματος αρχείου χρειάζεται να διαβάσει ένα i-node
 - ▶ πρώτα ελέγχει τον πίνακα των inodes
 - ▶ αν βρεθεί αυξάνει τον μετρητή και επιστρέφει έναν δείκτη στο i-node
 - ▶ αν δεν υπάρχει το i-node το μεταφέρει από την συσκευή E/E χρησιμοποιώντας τη μέθοδο `rw_inode`



Λειτουργίες Διαχείρισης i-Node (2)

- ▶ `put_inode`: όταν ολοκληρωθεί η συνάρτηση που χρησιμοποιήσε την `get_inode` το `inode` "επιστρέφεται" με αυτή την συνάρτηση
 - ▶ μειώνει τον μετρητή `i_count`
 - ▶ αν ο μετρητής πάρει την τιμή 0 τότε το αρχείο που περιγράφει το `i-node` δεν χρησιμοποιείται
 - ▶ αν χρειαστεί μπορούμε να αφαιρέσουμε το `i-node` από τον πίνακα
 - ▶ αν το `i-node` έχει αλλαγές (λέμε ότι είναι "dirty") μεταφέρεται στην μονάδα E/E
 - ▶ αν το πεδίο `i_link` είναι 0 τότε το αρχείο δεν περιέχεται σε κάποιο φάκελο – άρα μπορούμε να απελευθερώσουμε όλες τις ζώνες του αρχείου
 - ▶ Η μείωση του μετρητή `i_count` έχει διαφορετικές επιπτώσεις από την μείωση του μετρητή `i_link`



Λειτουργίες Διαχείρισης i-Node (3)

- ▶ `alloc_inode`: Δέσμευση ενός `i-node` όταν δημιουργείται ένα αρχείο
 - ▶ ελέγχει το `superblock` κατά πόσο η συσκευή επιτρέπει εγγραφή
 - ▶ ελέγχει το πεδίο `s_isearch` του `superblock`
 - ▶ σε αντίθεση με την `alloc_block` δεν προσπαθεί να βρει ένα κενό `i-node` κοντά σε κάποιο άλλο
- ▶ `free_inode`: Αποδέσμευση `i-node` (κατά την διαγραφή αρχείου)
 1. ενημερώνει το `superblock` (πεδίο `s_isearch`)
 2. ενημερώνει την αντίστοιχη θέση του `bitmap` θέτοντας την σε 0
- ▶ `update_times`: Ενημερώνει τα πεδία ημερομηνίας - ώρας του `i-node`
 - ▶ επικοινωνεί με τον πυρήνα – εκεί υλοποιούνται οι υπηρεσίες ρολογιού
 - ▶ χρησιμοποιείται από τις κλήσεις συστήματος `STAT` και `FSTAT`



Λειτουργίες Διαχείρισης i-Node (4)

- ▶ `rw_inode`: Μεταφορά `i-node` από/προς αποθηκευτική μονάδα από/προς `cache`
 - ▶ αντίστοιχη με την `rw_block`
 - ▶ υπολογίζει σε ποιο `block` είναι αποθηκευμένο το `i-node`
 - ▶ διαβάζει το `block` από την μονάδα E/E
 - ▶ εντοπίζει το `i-node` μέσα στο `block`
 - ▶ διαβάζει το `i-node` και το τοποθετεί στον πίνακα των `inodes`
 - ▶ καλεί την `put_block`
- ▶ `old_icopy`: Χρησιμοποιείται για την μετατροπή των `i-nodes` της έκδοσης V1 του συστήματος
- ▶ `new_icopy`: Χρησιμοποιείται για την μετατροπή των `i-nodes` της έκδοσης V2 του συστήματος
- ▶ `dup_icopy`: Αυξάνει τον μετρητή του `i-node` κάθε φορά που καλείται η `OPEN`



Λειτουργίες Διαχείρισης Superblock (1)

- ▶ Η δομή του `superblock` ορίζεται στο αρχείο `/usr/src/servers/fs/super.h`
- ▶ Οι συναρτήσεις που διαχειρίζονται το `superblock` ορίζονται στο αρχείο `/usr/src/servers/fs/super.c`
- ▶ 6 βασικές συναρτήσεις
- ▶ `alloc_bit`: χρησιμοποιείται όταν καλούμε την `alloc_inode` ή την `alloc_zone`
 - ▶ διαβάζει το αντίστοιχο `bitmap` για να εντοπίσει μια κενή θέση
 - ▶ πρόκειται για ένα `loop 3` επιπέδων
 - ▶ Το πρώτο επίπεδο ανατρέχει στα `blocks` όπου έχει αποθηκευτεί το `bitmap`
 - ▶ Το δεύτερο επίπεδο ανατρέχει στα `words` του κάθε `block`
 - ▶ Το τρίτο επίπεδο ανατρέχει τα `bit` του κάθε `word`



Λειτουργίες Διαχείρισης Superblock (2)

- ▶ `free_bit`: χρησιμοποιείται όταν καλούμε την `free_inode` ή την `free_zone`
 - ▶ πιο απλή από την `alloc_bit`
 - ▶ Εντοπίζει το block που περιέχει τη θέση του πίνακα
 - ▶ Ανατρέπει στο word και θέτει το αντίστοιχο bit σε 0
 - ▶ Για να διαβάσει το block καλεί την `get_block`
 - ▶ Για να αποθηκεύσει την αλλαγή καλεί την `put_block`
- ▶ `get_super`: ελέγχει τον πίνακα με τα superblock και επιστρέφει αυτό που αντιστοιχεί στην συγκεκριμένη συσκευή Ε/Ε
 - ▶ Την χρησιμοποιούμε για να ελέγξουμε κατά πόσο η συσκευή έχει ήδη γίνει mount
- ▶ `get_block_size`: εντοπίζει το μέγεθος των block στην συγκεκριμένη συσκευή Ε/Ε
 - ▶ Εντοπίζει το superblock της συσκευής και διαβάζει το συγκεκριμένο πεδίο



Λειτουργίες Διαχείρισης Superblock (3)

- ▶ `mounted`: αναφέρει κατά πόσο ένα συγκεκριμένο i-node βρίσκεται σε μια mounted συσκευή Ε/Ε
 - ▶ χρησιμοποιείται μόνο όταν μια συσκευή είναι κλειστή
 - ▶ Επιστρέφει TRUE αν η συσκευή είναι η ρίζα του συστήματος αρχείων
 - ▶ ή αν η συσκευή είναι mounted
- ▶ `read_super`: διαβάζει το superblock από μια συσκευή Ε/Ε
 - ▶ ανάλογη με την `read_block` και `read_inode`
 - ▶ δεν διαβάζει από την προσωρινή μνήμη – διαβάζει απ' ευθείας από την συσκευή Ε/Ε
 - ▶ ελέγχει την έκδοση του συστήματος αρχείων της συσκευής και κάνει τις απαραίτητες μετατροπές στην δομή superblock αμέσως μετά την ανάγνωση
 - ▶ ανεξάρτητα από την έκδοση του συστήματος αρχείων που έχει κάθε συσκευή Ε/Ε, τα superblock που διατηρούνται στην μνήμη έχουν την ίδια δομή



Λειτουργίες Διαχείρισης Πίνακα Ανοικτών Αρχείων (1)

- ▶ Η δομή του πίνακα των ανοικτών αρχείων ορίζεται στο αρχείο `/usr/src/servers/fs/file.h`

```
EXTERN struct filp {
    mode_t filp_mode; /* RW bits - how file is opened
    int filp_flags; /* flags from open and fcntl */
    int filp_count; /* file descriptors share this sl
    struct inode *filp_ino; /* pointer to the inode *
    off_t filp_pos; /* file position */

    /* the following fields are for select() */
    int filp_selectors; /* select()ing processes bloc
    int filp_select_ops; /* interested in these SEL_*
    int filp_pipe_select_ops; /* fd-type-specific se
} filp[NR_FILPS];
```



Λειτουργίες Διαχείρισης Πίνακα Ανοικτών Αρχείων (2)

- ▶ Οι συναρτήσεις που διαχειρίζονται τον πίνακα των ανοικτών αρχείων ορίζονται στο αρχείο `/usr/src/servers/fs/filedes.c`
- ▶ 4 βασικές συναρτήσεις
- ▶ `get_fd`: επιστρέφει ένα νέο file descriptor ενός συγκεκριμένου αρχείου
 - ▶ ανατρέπει τον πίνακα και εντοπίζει μια κενή εγγραφή
 - ▶ χρησιμοποιείται από τις κλήσεις συστήματος OPEN και CREAT
- ▶ `get_filp`: εντοπίζει ένα file descriptor
 - ▶ ανατρέπει τον πίνακα και εντοπίζει το file descriptor
- ▶ `find_filp`: εντοπίζει όλες τις διεργασίες που έχουν ένα file descriptor για ένα συγκεκριμένο αρχείο
- ▶ `inval_filp`: κλείνει ένα file descriptor



Λειτουργίες Διαχείρισης File Locks (1)

- ▶ Το MINIX 3 επιτρέπει να κάνουμε lock ένα μέρος ενός αρχείου για ανάγνωση, εγγραφή ή και τα δύο
- ▶ Το κλείδωμα ενός (μέρους) αρχείου γίνεται με την κλήση συστήματος FCNTL
- ▶ Η δομή του πίνακα των file locks ορίζεται στο αρχείο `/usr/src/servers/fs/lock.h`

```
EXTERN struct file_lock {  
    short lock_type; /* F_RDLOCK or F_WRLOCK; 0 means  
    pid_t lock_pid; /* pid of the process holding the  
    struct inode *lock_inode; /* pointer to the inode  
    off_t lock_first; /* offset of first byte locked  
    off_t lock_last; /* offset of last byte locked */  
} file_lock[NR_LOCKS];
```



Λειτουργίες Διαχείρισης File Locks (2)

- ▶ Η κλήση συστήματος FCNTL προσφέρει 3 λειτουργίες σχετικά με το κλείδωμα μέρους ενός αρχείου
 1. F_SETLK – Κλειδώνει μια περιοχή ενός αρχείου για ανάγνωση και εγγραφή
 2. F_SETLKW – Κλειδώνει μια περιοχή ενός αρχείου για εγγραφή
 3. F_GETLK – Ελέγχει αν μια περιοχή ενός αρχείου είναι κλειδωμένη
- ▶ Οι συναρτήσεις που διαχειρίζονται τον πίνακα των file locks ορίζονται στο αρχείο `/usr/src/servers/fs/lock.c`
- ▶ 4 βασικές συναρτήσεις
- ▶ `lock_op`: καλείται όταν θέλουμε να κλειδώσουμε ένα (μέρος) αρχείου με την κλήση συστήματος FCNTL
 - ▶ ελέγχει αν το μέρος του αρχείου που θέλουμε να κλειδώσουμε υπάρχει
 - ▶ ελέγχει αν το νέο κλείδωμα καλύπτει περιοχές που είναι ήδη κλειδωμένες



Λειτουργίες Διαχείρισης File Locks (3)

- ▶ `lock_revive`: καλείται όταν θέλουμε να ξεκλειδώσουμε ένα μέρος ενός αρχείου
 - ▶ ενεργοποιεί όλες τις διεργασίες που περιμένουν να διαβάσουν το αρχείο (ή το μέρος του αρχείου)
 - ▶ ο εντοπισμός των διεργασιών δεν είναι εύκολος
 - ▶ για να αποφύγουμε σύνθετα τμήματα κώδικα, υποθέτουμε ότι το κλείδωμα των αρχείων δεν είναι σύνθετος
 - ▶ στην ουσία οι διεργασίες που έχουν μπλοκαριστεί, περιοδικά ξανά-τρέχουν. Αν το αρχείο είναι ακόμα κλειδωμένο, θα μπλοκάρουν ξανά
 - ▶ δεν είναι αποδοτικός τρόπος για να υλοποιήσουμε βάσεις δεδομένων (για πολλούς χρήστες)
 - ▶ η μέθοδος καλείται και όταν θέλουμε να κλείσουμε ένα αρχείο – για να διαγραφούν τα κλειδώματα



Σύνοψη 13^{ης} Διάλεξης

Προηγούμενο Μάθημα

Σύστημα Αρχείων
Δομή Συστήματος Αρχείων
Ανάγνωση Αρχείων

Λειτουργικό Σύστημα Minix

Διαχείριση Block
Διαχείριση I-Node
Διαχείριση Superblock, Πίνακα Ανοικτών Αρχείων, File Locks

Σύνοψη Μαθήματος

Σύνοψη Μαθήματος
Βιβλιογραφία
Επόμενη Διάλεξη



- ▶ Σύστημα Αρχείων στο Λ.Σ. MINIX 3
- ▶ Εσωτερική δομή συστήματος αρχείων στο Λ.Σ. MINIX 3
- ▶ Διαχείριση Block
- ▶ Διαχείριση i-Node
- ▶ Διαχείριση Superblock
- ▶ Διαχείριση Πίνακα Ανοικτών Αρχείων
- ▶ Διαχείριση File Lock



- ▶ Βιβλίο "Operating Systems Design and Implementation, Third Edition" (Andrew S. Tanenbaum)
 1. Κεφάλαιο 5: File System
 - ▶ Παράγραφος 5.6 Overview of the MINIX 3 File System
 - ▶ Παράγραφος 5.7 Implementation of the MINIX 3 File System
- ▶ Βιβλίο "Σύγχρονα Λειτουργικά Συστήματα" (A.Tanenbaum)
 1. Κεφάλαιο 6: Συστήματα Αρχείων
 - ▶ Παράγραφος 6.3 Υλοποίηση Συστήματος Αρχείων
 2. Κεφάλαιο 10: Μελέτη Περίπτωσης 1: Unix και Linux
 - ▶ Παράγραφος 10.6 Το Σύστημα Αρχείων του Unix
 - ▶ Παράγραφος 10.6.1 Θεμελιώδεις έννοιες



- ▶ Σύστημα Αρχείων στο Λ.Σ. MINIX 3
- ▶ Βιβλίο "Operating Systems Design and Implementation, Third Edition" (Andrew S. Tanenbaum)
 1. Κεφάλαιο 5: Συστήματα Αρχείων
 - ▶ Παράγραφος 5.6 Overview of the MINIX 3 File System
 - ▶ Παράγραφος 5.7 Implementation of the MINIX 3 File System
- ▶ Βιβλίο "Σύγχρονα Λειτουργικά Συστήματα" (A.Tanenbaum)
 1. Κεφάλαιο 6: Συστήματα Αρχείων
 - ▶ Παράγραφος 6.3 Υλοποίηση Συστήματος Αρχείων
 2. Κεφάλαιο 10: Μελέτη Περίπτωσης 1: Unix και Linux
 - ▶ Παράγραφος 10.6 Το Σύστημα Αρχείων του Unix
 - ▶ Παράγραφος 10.6.1 Θεμελιώδεις έννοιες

