

Εργαστήριο Λειτουργικών Συστημάτων

Μάθημα 6^{ου} Εξαμήνου,

Τομέας Λογικού και Υπολογιστών

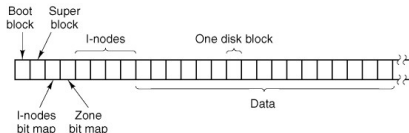
Ιωάννης Χατζηγιαννάκης

Τετάρτη 26 Μαΐου 2010
Αίθουσα ΒΑ



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

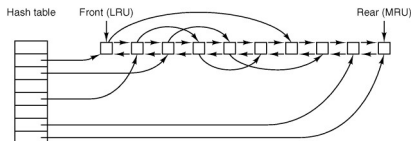


Διαχείριση Block

- ▶ 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`, `rear`



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



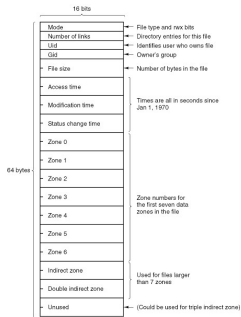
Λειτουργίες Διαχείρισης Block

- ▶ Οι συναρτήσεις που διαχειρίζονται τις δομές της προσωρινής μνήμης block ορίζονται στο αρχείο `/usr/src/servers/fs/cache.h`
- ▶ `get_block`: επιστρέφει ένα block
- ▶ `put_block`: τοποθετεί ένα block στην προσωρινή μνήμη
- ▶ `alloc_zone`: Δέσμευση νέας ζώνης
- ▶ `free_zone`: Αποδέσμευση ζώνης
- ▶ `rw_block`: Μεταφορά block από/προς αποθηκευτική μονάδα προς/από cache
- ▶ `invalidate`: Άδεια προσωρινής μνήμης
- ▶ `flushall`: Αποθήκευση όλων των block που έχουν αλλαγές
- ▶ `rw_scattered`: Μεταφορά πολλαπλών block από/προς αποθηκευτική μονάδα προς/από cache
- ▶ `rm_lru`: Διαγραφή του πρώτου block από την λίστα LRU

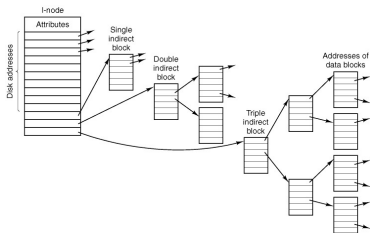


Το i-node του MINIX 3

- ▶ Τα inodes 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



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
- ▶ `put_inode`: όταν ολοκληρωθεί η συνάρτηση που χρησιμοποίησε την `get_inode` το i-node 'επιστρέφεται' με αυτή την συνάρτηση
- ▶ `alloc_inode`: Δέσμευση ενός i-node όταν δημιουργείται ένα αρχείο



Λειτουργίες Διαχείρισης i-Node (2)

- ▶ `free_inode`: Αποδέσμευση i-node (κατά την καταγραφή αρχείου)
- ▶ `update_times`: Ενημερώνει τα πεδία ημερ/νίας - ώρας του i-node
- ▶ `rw_inode`: Μεταφορά i-node από/προς αποθηκευτική μονάδα προς/από cache
- ▶ `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



Το superblock του MINIX 3

- ▶ Το superblock περιέχει πληροφορίες για τη δομή του συστήματος αρχείου
- ▶ Έχει πάντα μέγεθος 1024 bytes
- ▶ Βάση του πλήθους των i-nodes και το μέγεθος των blocks μπορούμε να υπολογίσουμε το μέγεθος του bitmap των inodes
- ▶ Ο αποθηκευτικός χώρος διαίρεται 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

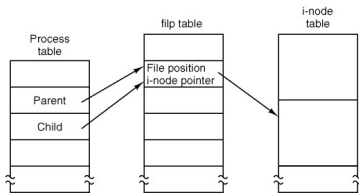


Λειτουργίες Διαχείρισης Superblock

- ▶ 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`
- ▶ `alloc_bit`: χρησιμοποιείται όταν καλούμε την `alloc_inode` ή την `alloc_zone`
- ▶ `free_bit`: χρησιμοποιείται όταν καλούμε την `free_inode` ή την `free_zone`
- ▶ `get_super`: ελέγχει τον πίνακα με τα superblock και επιστρέφει αυτό που αντιστοιχεί στην συγκεκριμένη συσκευή E/E
- ▶ `get_block_size`: εντοπίζει το μέγεθος των block στην συγκεκριμένη συσκευή E/E
- ▶ `mounted`: αναφέρει κατά πόσο ένα συγκεκριμένο i-node βρίσκεται σε μια mounted συσκευή E/E
- ▶ `read_super`: διαβάζει το superblock από μια συσκευή E/E



Πίνακας Ανοικτών Αρχείων Θέσεων



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



Λειτουργίες Διαχείρισης Πίνακα Ανοικτών Αρχείων

- ▶ 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
- ▶ `get_filp`: εντοπίζει ένα 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

- ▶ Το MINIX 3 επιτρέπει να κάνουμε lock ένα μέρος ενός αρχείου για ανάγνωση, εγγραφή ή και τα δύο
- ▶ 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
- ▶ Η δομή του πίνακα των file locks ορίζεται στο αρχείο `/usr/src/servers/fs/lock.h`
- ▶ Οι συναρτήσεις που διαχειρίζονται τον πίνακα των file locks ορίζονται στο αρχείο `/usr/src/servers/fs/lock.c`
- ▶ `lock_op`: κλειδώνει ένα (μέρος) αρχείου με την κλήση συστήματος FCNTL
- ▶ `lock_revive`: ξεκλειδώνει ένα μέρος ενός αρχείου



Σύνοψη 14^{ης} Διάλεξης

Προηγούμενο Μάθημα

Διαχείριση Block

Διαχείριση I-Node

Διαχείριση Superblock, Πίνακα Ανοικτών Αρχείων, File Locks

Λειτουργικό Σύστημα Minix

Αρχειοποίηση Συστήματος Αρχείων

Άνοιγμα, Δημιουργία Ανάγνωση Αρχείων

Συσκευές Εισόδου / Εξόδου

Σύνοψη Μαθήματος

Σύνοψη Μαθήματος

Βιβλιογραφία



Αρχειοποίηση Συστήματος Αρχείων

- ▶ Η κεντρική συνάρτηση του συστήματος αρχείων ορίζεται στο αρχείο `/usr/src/servers/fs/main.c`
- ▶ Η αρχειοποίηση του συστήματος γίνεται από την συνάρτηση `fs_init`
- ▶ Το πρώτο βήμα της αρχειοποίησης είναι η δημιουργία του πίνακα των διεργασιών
 - ▶ Το σύστημα αρχείων αρχειοποιεί τον πίνακα των διεργασιών σε συνεργασία με τον Διαχειριστή Διεργασιών
 - ▶ Ο Διαχειριστής Διεργασιών στέλνει τις εγγραφές του πίνακα υπο την μορφή μηνυμάτων
 - ▶ Για κάθε διεργασία στέλνει ένα μήνυμα με το PID και τα UID, GID
 - ▶ Αμέσως μετά το μήνυμα που περιγράφει την τελευταία διεργασία, ο Διαχειριστής Διεργασιών στέλνει ένα τελευταίο μήνυμα για να ενημερώσει ότι δεν υπάρχουν άλλες διεργασίες
 - ▶ Το Σύστημα Αρχείων απαντάει στέλνοντας ένα μήνυμα OK



Αρχικοποίηση Πίνακα Διεργασιών

```
do {
    if (OK != (s=receive(PM_PROC_NR, &mess)))
        panic(__FILE__, "FS couldn't receive from PM", s);
    if (NONE == mess.PR_ENDPT) break;
    rfp = &fproc[mess.PR_SLOT];
    rfp->fp_pid = mess.PR_PID;
    rfp->fp_endpoint = mess.PR_ENDPT;
    rfp->fp_realuid = (uid_t) SYS_UID;
    rfp->fp_effuid = (uid_t) SYS_UID;
    rfp->fp_realgid = (gid_t) SYS_GID;
    rfp->fp_effgid = (gid_t) SYS_GID;
    rfp->fp_umask = ~0;
} while (TRUE);
mess.m_type = OK;
s=send(PM_PROC_NR, &mess);
```



Εσωτερικοί Έλεγχοι

- ▶ Αμέσως μετά την αρχικοποίηση του πίνακα διεργασιών γίνονται ορισμένοι εσωτερικοί έλεγχοι

```
if (OPEN_MAX > 127)
    panic(__FILE__, "OPEN_MAX > 127", NO_NUM);
if (NR_BUFS < 6)
    panic(__FILE__, "NR_BUFS < 6", NO_NUM);
if (V1_INODE_SIZE != 32)
    panic(__FILE__, "V1 inode size != 32", NO_NUM);
if (V2_INODE_SIZE != 64)
    panic(__FILE__, "V2 inode size != 64", NO_NUM);
if (OPEN_MAX > 8 * sizeof(long))
    panic(__FILE__, "Too few bits in fp_cloexec", NO_NUM);
```



Αρχικοποίηση Προσωρινής Μνήμης (1)

- ▶ Μετά απο τους εσωτερικούς ελέγχους γίνεται η αρχικοποίηση της προσωρινής μνήμης
- ▶ Χρησιμοποιείται η συνάρτηση buf_pool()
 - ▶ Ορίζεται στο αρχείο main.c
- ▶ Αρχικοποιεί την συνδεδεμένη λίστα των block
- ▶ Αρχικοποιεί την πίνακα με τις τιμές hash
- ▶ Αρχικοποιεί τις δύο ουρές (front, rear)

```
bufs_in_use = 0;
front = &buf[0];
rear = &buf[NR_BUFS - 1];
```



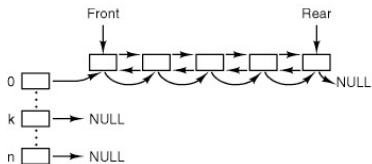
Αρχικοποίηση Προσωρινής Μνήμης (2)

```
for (bp = &buf[0]; bp < &buf[NR_BUFS]; bp++) {
    bp->b_blocknr = NO_BLOCK;
    bp->b_dev = NO_DEV;
    bp->b_next = bp + 1;
    bp->b_prev = bp - 1;
}
buf[0].b_prev = NIL_BUF;
buf[NR_BUFS - 1].b_next = NIL_BUF;

for (bp = &buf[0]; bp < &buf[NR_BUFS]; bp++)
    bp->b_hash = bp->b_next;
buf_hash[0] = front;
```



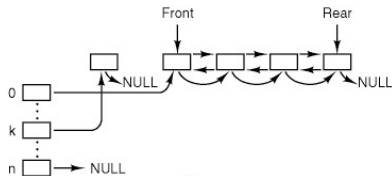
Παράδειγμα Προσωρινής Μνήμης



- ▶ Όλα τα block βρίσκονται στις δύο ουρές LRU, MRU
- ▶ Όλα τα block είναι τοποθετημένα στην πρώτη ουρά hash
- ▶ Όλα τα block σχηματίζουν την συνδεδεμένη λίστα (δύο κατευθύνσεων)



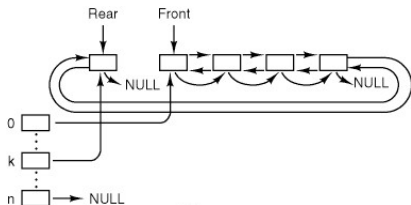
Χρήση ενός block από την Προσωρινή Μνήμη



- ▶ Όταν ζητηθεί ένα block για χρήση με την `get_block`
- ▶ Το πρώτο διαθέσιμο block της LRU αφαιρείται
- ▶ Αντιγράφουμε σε αυτό την τιμή από την μονάδα E/E
- ▶ Τοποθετείτε στην σωστή ουρά hash



Επιστροφή ενός block στην Προσωρινή Μνήμη



- ▶ Όταν επισταφεί το block αμέσως μετά την χρήση του με την `put_block`
- ▶ Παραμένει στις ουρές για μελλοντική χρήση



Ολοκλήρωση Αρχικοποίηση Συστήματος Αρχείων

- ▶ Μετά από την αρχικοποίηση της προσωρινής μνήμης
- ▶ Γίνεται σύνδεση/αναγνώριση των συσκευών E/E με την χρήση της συνάρτησης `build_dmap()`
- ▶ Γίνεται αρχικοποίηση του ramdisk με την χρήση της συνάρτησης `init_root()`
 - ▶ Χρησιμοποιείται από το υπόλοιπο ΛΣ σε διάφορες λειτουργίες
- ▶ Γίνεται ανάγνωση του superblock της κεντρικής συσκευής E/E με την χρήση της συνάρτησης `init_root()`



Δημιουργία, Άνοιγμα Κλείσιμο Αρχείων

- ▶ Οι λειτουργίες που υλοποιούν το άνοιγμα των αρχείων για χρήση βρίσκονται στο αρχείο `open.c`
- ▶ Πρόκειται για 6 κλήσεις συστήματος
 1. CREAT
 2. OPEN
 3. MKNOD
 4. MKDIR
 5. CLOSE
 6. LSEEK
- ▶ Σε παλαιότερες εκδόσεις συστημάτων UNIX οι κλήσεις συστήματος CREAT και OPEN ήταν διαφορετικές
- ▶ Πλέον δεν υπάρχει λόγος:
 - ▶ Αν το αρχείο που πρέπει να ανοίξει δεν υπάρχει, δημιουργείται καινούργιο

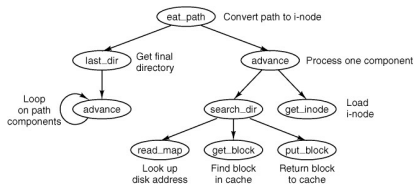


Μετατροπή Path σε i-node

- ▶ Ορισμένες κλήσεις συστήματος εκτελούνται βάση ενός path που υποδηλώνει την θέση του αρχείου στο σύστημα
- ▶ Οι κλήσεις του συστήματος πρέπει να εντοπίσουν το i-node που αντιστοιχεί στο αρχείο
- ▶ Η υλοποίηση της μετατροπής του path στο αντίστοιχο i-node γίνεται στο αρχείο `path.c`
- ▶ Η βασική συνάρτηση είναι η `eat_path()`
 - ▶ Ξεκινάει από τον τελικό φάκελο που ορίζει το path χρησιμοποιώντας την συνάρτηση `last_dir`
 - ▶ Εξετάζει το path αναδρομικά με την χρήση της συνάρτησης `advance`
 - ▶ Αν δεν εντοπίσει το i-node τότε επιστρέφει το `NIL_INODE`
 - ▶ Αν εντοπιστεί το i-node το τοποθετεί στον πίνακα των i-node



Μετατροπή Path σε i-node

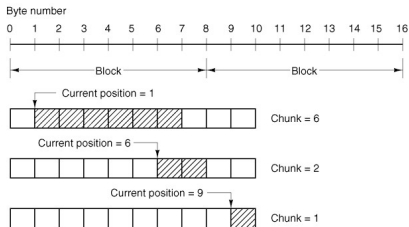


Ανάγνωση Αρχείων (1)

- ▶ Εφόσον έχουμε ανοίξει ένα αρχείο μπορούμε να διαβάσουμε τα δεδομένα
- ▶ Η υλοποίηση της ανάγνωσης αρχείων γίνεται στο αρχείο `read.c`
- ▶ Γίνονται βασικοί έλεγχοι για τις παραμέτρους που έχει κληθεί η λειτουργία του συστήματος
- ▶ Αρχικοποιούνται οι μεταβλητές όπου θα αποθηκευτούν τα δεδομένα που διαβάσθηκαν
- ▶ Στην συναίχεια ξεκινάει ένας βρόγχος για να διαβαστούν όλα τα block που περιέχουν τα δεδομένα που ζητήθηκαν
- ▶ Τα δεδομένα διαβάζονται σε chunks
 - ▶ Κάθε chunk χωράει σε ένα disk block
- ▶ Ο βρόγχος συναίχεται έως ότου
 1. Αναγνωστούν όλα τα δεδομένα που ζητήθηκαν
 2. Έχει διαβαστεί όλο το block
 3. Τελειώσει το αρχείο
- ▶ Η βασική λειτουργία της ανάγνωσης ενός chunk γίνεται με την χρήση της `rw_chunk`



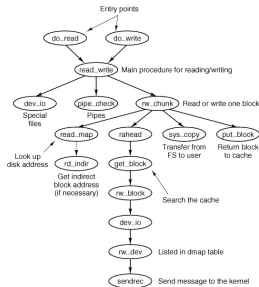
Ανάγνωση Αρχείων (2)



- ▶ Παράδειγμα για την ανάγνωση chunk από ένα αρχείο 10 byte όταν ζητάμε να διαβάσουμε 6 bytes



Ανάγνωση Αρχείων (3)



Εγγραφή Αρχείων

- (a) 24 Free zones: 12 20 31 36...
- (b) 24 25
- (c) 24 25 40
- (d) 24 25 40 41
- (e) 24 25 40 41 62
- (f) 24 25 40 41 62 63
- Block number

- ▶ Ανάθεση block σε ένα αρχείο, όταν το ΣΑ έχει 1 Kbyte blocks και 2 Kbyte zones

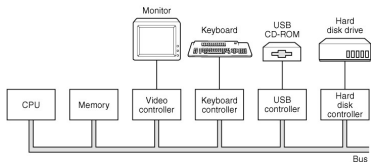


Γενικά Θέματα συσκευών εισόδου / εξόδου

- ▶ Κάθε άνθρωπος αντιλαμβάνεται τις συσκευές εισόδου / εξόδου με διαφορετικό τρόπο
 - ▶ Ο ηλεκτρολόγος μηχανικός βλέπει σε επίπεδο υλικού
 - ▶ Ο προγραμματιστής βλέπει σε επίπεδο διαπαφής με το λογισμικό
 - ▶ Ο χρήστης βλέπει σε επίπεδο εφαρμογής
 - ▶ ...
- ▶ Υπάρχουν διαφορετικών ειδών συσκευές - χωρίζονται (χοντρικά) σε δύο βασικές κατηγορίες
 - ▶ Μηλοκ συσκευές (block devices)
 - ▶ Συσκευές χαρακτήρων (character devices)
- ▶ Ο χειρισμός μιας συσκευής σε επίπεδο υλικού γίνεται με την χρήση ενός ελεγκτή (controller) ή προσαρμογέα (adaptor)



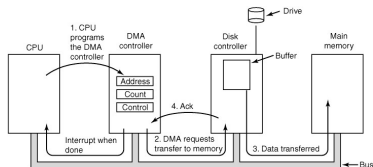
Συνδεσμολογία συσκευών εισόδου / εξόδου



- ▶ Η επικοινωνία των συσκευών με τους ελεγκτές γίνεται σε χαμηλό επίπεδο
- ▶ Οι ελεγκτές επικοινωνούν με τον επεξεργαστή με την χρήση interrupt (και ορισμένους registers)
- ▶ Η ανταλλαγή πληροφορίας γίνεται με την χρήση της μνήμης



Direct Memory Access (DMA)



- ▶ Είναι απαραίτητο ο επεξεργαστής να επικοινωνεί με τον ελεγκτή για την μεταφορά δεδομένων
- ▶ Η μεταφορά της πληροφορίας γίνεται με 1 byte κάθε φορά – αυτό καθυστερεί υπερβολικά το σύστημα
- ▶ Χρησιμοποιούμε τον ελεγκτή DMA για να επιταχύνουμε διαδικασία



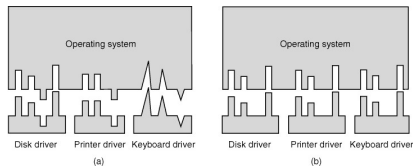
Αρχές Λογισμικού Εισόδου / Εξόδου

- ▶ Βασική αρχή για τον σχεδιασμό είναι η ανεξαρτησία του λογισμικού από την τεχνολογία του υλικού της συσκευής
- ▶ Θέλουμε να αναπτύξουμε εφαρμογές χωρίς να πρέπει να ορίσουμε εκ των προτέρων την συσκευή που θα συνδέσουμε στο τελικό σύστημα
- ▶ Χωρίς να περιοριζόμαστε σε συγκεκριμένο κατασκευαστή
- ▶ Ο χειρισμός των λαθών να είναι ανεξάρτητος του τύπου συσκευής
- ▶ Να μην γράψουμε επιπλέον κώδικα για να χειριστούμε συσκευές περιορισμένων δυνατοτήτων (π.χ. μικρή/αργή μνήμη)
- ▶ Ιεραρχούμε το σύστημα που σχετίζεται με τις συσκευές εισόδου / εξόδου σε 5 επίπεδα

1. Υλικό συσκευής
2. Χειριστές Interrupt
3. Οδηγοί Συσκευών (device drivers)
4. Διαχειριστές Συσκευών (ανεξάρτητοι από τους οδηγούς)
5. Λογισμικό Εισόδου / Εξόδου (user space)



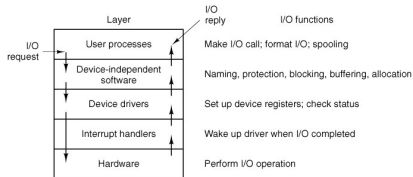
Διαχειριστές Συσκευών (ανεξάρτητοι από τους οδηγούς)



- ▶ Ενιαία διεπαφή χειρισμού/επικοινωνίας συσκευών
- ▶ Ενιαίος τρόπος χειρισμός λαθών
- ▶ Ενιαίος τρόπος απαρίθμησης / ονομασίας συσκευών



Ροή Πρόσβασης στις Συσκευές Εισόδου / Εξόδου



- ▶ Η ροή εκτέλεσης μιας αίτησης εισόδου / εξόδου
- ▶ π.χ. για την εκτύπωση ενός μηνύματος στην οθόνη



Οδηγοί Συσκευών Εισόδου / Εξόδου στο Minix (1)

- ▶ Για κάθε τύπο συσκευής ΕΕ υπάρχει ένας ανεξάρτητος οδηγός (device driver)
- ▶ Οι οδηγοί συσκευών είναι κανονικές διεργασίες
- ▶ Οι οδηγοί επικοινωνούν με το σύστημα αρχείων με την χρήση του μηχανισμού ανταλλαγής μηνυμάτων
- ▶ Ο κώδικας των οδηγών βρίσκεται στον φάκελο `/usr/src/drivers`
- ▶ Ένας απλός οδηγός υλοποιείται σε ένα μόνο αρχείο, π.χ. ο οδηγός του εκτυπωτή υλοποιείται στο αρχείο `/usr/src/drivers/printer/printer.c`
- ▶ Για τους οδηγούς RAM disk, hard disk και floppy disk υπάρχει ένα αρχείο για κάθε τύπο συσκευής και ένα κεντρικό αρχείο που περιέχει τις βασικές/κοινές συναρτήσεις (φάκελος `/usr/src/drivers/libdriver`, αρχεία `driver.c` και `drvlib.c`)
 - ▶ Πρόκειται για μια διεπαφή ανεξάρτητη από το τύπο της συσκευής



Οδηγοί Συσκευών Εισόδου / Εξόδου στο Minix (2)

- ▶ Ο κώδικας των οδηγών για το τερματικό οργανώνεται με ίδιο τρόπο
- ▶ Το αρχείο `/usr/src/drivers/tty/tty.c` υλοποιεί έναν οδηγό ανεξάρτητο από τον τύπο της συσκευής
- ▶ Για κάθε τύπο συσκευής υπάρχει ένα επιπλέον αρχείο που υλοποιεί τις λειτουργίες που εξαρτώνται από το υλικό
- ▶ Οι οδηγοί επικοινωνούν με το σύστημα αρχείων με την χρήση του μηχανισμού ανταλλαγής μηνυμάτων
 - ▶ Ο οδηγός της κονσόλας υλοποιείται στο αρχείο `/usr/src/drivers/tty/console.c`
 - ▶ Ο οδηγός του πληκτρολογίου υλοποιείται στο αρχείο `/usr/src/drivers/tty/keyboard.c`
 - ▶ Ο οδηγός της θύρας RS232 υλοποιείται στο αρχείο `/usr/src/drivers/tty/rs232.c`
- ▶ Στο αρχείο `tty.c` γίνεται ο έλεγχος των εισερχόμενων μηνυμάτων
- ▶ Στα αρχεία `console.c`, `keyboard.c`, `rs232.c` γίνεται ο χειρισμός της συσκευής

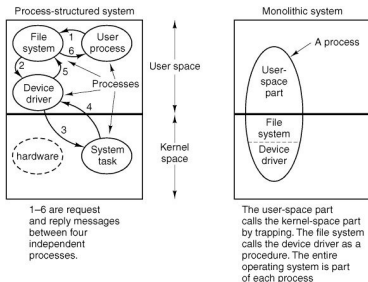


Θέματα Υλοποίησης Οδηγών Εισόδου / Εξόδου

- ▶ Ο τρόπος που υλοποιούνται οι οδηγοί των συσκευών ΕΕ στο Minix είναι χαρακτηριστικός
 - ▶ Διαφέρει ριζικά από τον τρόπο υλοποίησης άλλων UNIX/LINUX συστημάτων
 - ▶ Βασίζεται στην σχεδιαστική αρχή της εκτέλεσης των συνιστωσών του Λ.Σ. ως διαφορετικές διεργασίες στο user space
 - ▶ Είναι ιδιαίτερα ευέλικτος και επεκτάσιμος
 - ▶ Επιπυχνάνει μειωμένη απόδοση (σε σχέση με τα μονολιθικά συστήματα) λόγω της ανταλλαγής μηνυμάτων
- ▶ Μια διεργασία που θέλει να χρησιμοποιήσει μια συσκευή
 - ▶ Επικοινωνεί με το Σύστημα Αρχείων (αποστολή μηνύματος)
 - ▶ Το Σύστημα Αρχείων επικοινωνεί με τον Οδηγό της συσκευής (αποστολή μηνύματος)
 - ▶ Ο οδηγός επικοινωνεί με τον πυρήνα (αποστολή μηνύματος στο system task)
 - ▶ Ο πυρήνας επικοινωνεί με την συσκευή (επίπεδο υλικού)



2 Τρόποι Επικοινωνίας Χειριστών-Συστήματος



Σύνοψη 14^{ης} Διάλεξης

Προηγούμενο Μάθημα

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

Λειτουργικό Σύστημα Minix

- Αρχικοποίηση Συστήματος Αρχείων
- Άνοιγμα, Δημιουργία Ανάγνωση Αρχείων
- Συσκευές Εισόδου / Εξόδου

Σύνοψη Μαθήματος

- Σύνοψη Μαθήματος
- Βιβλιογραφία

Στο σημερινό μάθημα είδαμε

- ▶ Σύστημα Αρχείων στο Λ.Σ. MINIX 3
- ▶ Εσωτερική δομή συστήματος αρχείων στο Λ.Σ. MINIX 3
- ▶ Αρχικοποίηση Συστήματος Αρχείων
- ▶ Ανάγνωση Αρχείων
- ▶ Εγγραφή Αρχείων
- ▶ Συσκευές Εισόδου / Εξόδου
- ▶ Οδηγοί Συσκευών Εισόδου / Εξόδου

Βιβλιογραφία

- ▶ Βιβλίο "Operating Systems Design and Implementation, Third Edition" (Andrew S. Tanenbaum)
 1. Κεφάλαιο 3: Input / Output
 2. 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 5: Συσκευές Εισόδου / Εξόδου
 2. Κεφάλαιο 6: Συστήματα Αρχείων
 - ▶ Παράγραφος 6.3 Υλοποίηση Συστήματος Αρχείων
 3. Κεφάλαιο 10: Μελέτη Περίπτωσης 1: Unix και Linux
 - ▶ Παράγραφος 10.5 Είσοδος / Έξοδος στο UNIX
 - ▶ Παράγραφος 10.6 Το Σύστημα Αρχείων του Unix
 - ▶ Παράγραφος 10.6.1 Θεμελιώδεις έννοιες