

## Εργαστήριο Λειτουργικών Συστημάτων

Μάθημα 6<sup>ου</sup> Εξαμήνου,

Τομέας Λογικού και Υπολογιστών

Ιωάννης Χατζηγιαννάκης

Τετάρτη 12 Μαΐου 2010  
Αίθουσα ΒΑ



## Επικοινωνία Διεργασιών – Απ' ευθείας Αντιγραφή Μνήμης

- ▶ Ο πιο βασικός τρόπος είναι μέσω απ' ευθείας πρόσβαση στη μνήμη
- ▶ Ο πυρήνας παρέχει μηχανισμούς (κλήσεις πυρήνα) για την αντιγραφή μνήμης
  - ▶ Είναι αυτός ο τρόπος ασφαλής;
  - ▶ Αρχιτεκτονική Μικρο-πυρήνα – μόνο οι διαχειριστές / οδηγοί μπορούν να καλέσουν αυτούς τους μηχανισμούς
- ▶ Εικονικές Διευθύνσεις:
  - ▶ Οι διεργασίες χειρίζονται τα τμήματα μνήμης τους μέσω των εικονικών διευθύνσεων
  - ▶ Ο πυρήνας γνωρίζει τις πραγματικές διευθύνσεις που έχουν τα τμήματα των διεργασιών ...
  - ▶ Μετατρέπει τις εικονικές διευθύνσεις σε πραγματικές



## Αντιγραφή Μνήμης – Κλήσεις Πυρήνα – SYS\_VIRCOPY

```
_PROTOTYPE(int sys_vircopy, (int src_proc,  
int src_seg, vir_bytes src_vir,  
int dst_proc, int dst_seg, vir_bytes dst_vir,  
phys_bytes bytes));
```

- ▶ Αντιγραφή από – διεργασία 'πηγή' (source)
  - ▶ src\_proc -- θέση διεργασίας στον πίνακα διεργασιών
  - ▶ src\_seg -- τμήμα μνήμης διεργασίας (text, data, stack)
  - ▶ src\_vir -- εικονική θέση μνήμης στη διεργασία
- ▶ Αντιγραφή προς – διεργασία 'προορισμός' (destination)
  - ▶ dst\_proc -- θέση διεργασίας στον πίνακα διεργασιών
  - ▶ dst\_seg -- τμήμα μνήμης διεργασίας (text, data, stack)
  - ▶ dst\_vir -- εικονική θέση μνήμης στη διεργασία
- ▶ bytes -- Πλήθος bytes προς αντιγραφή



## Αντιγραφή Μνήμης – Κλήσεις Πυρήνα

- ▶ Στο αρχείο /usr/src/kernel/system.h ορίζονται οι handlers για τις κλήσεις πυρήνα

```
_PROTOTYPE( int do_copy, (message *m_ptr) );  
#define do_vircopy do_copy  
#define do_physcopy do_copy
```

- ▶ Στο αρχείο /usr/src/kernel/system.c γίνεται η αντιστοίχιση των handlers με τους αριθμούς των κλήσεων πυρήνα

```
map(SYS_VIRCOPY, do_vircopy);  
map(SYS_PHYSCOPY, do_physcopy);
```



## Συνάρτηση – do\_copy (1)

- ▶ Μετατρέπει τις παραμέτρους του μηνύματος σε 2 μεταβλητές τύπου vir\_addr

```
PUBLIC int do_copy(m_ptr)
register message *m_ptr;
{
    struct vir_addr vir_addr[2];
    phys_bytes bytes;

    vir_addr[_SRC_].proc_nr_e = m_ptr->CP_SRC_ENDPT;
    vir_addr[_SRC_].segment = m_ptr->CP_SRC_SPACE;
    vir_addr[_SRC_].offset = (vir_bytes) m_ptr->CP_SRC_ADD;
    vir_addr[_DST_].proc_nr_e = m_ptr->CP_DST_ENDPT;
    vir_addr[_DST_].segment = m_ptr->CP_DST_SPACE;
    vir_addr[_DST_].offset = (vir_bytes) m_ptr->CP_DST_ADD;
    bytes = (phys_bytes) m_ptr->CP_NR_BYTES;
```



## Συνάρτηση – do\_copy (2)

- ▶ Ελέγχει τις διευθύνσεις – αν αναφέρονται σε ‘πραγματικές’ διεργασίες
  - ▶ Χρησιμοποιεί την συνάρτηση isokendpt\_f
  - ▶ ... ορίζεται στο αρχείο /usr/src/kernel/proc.c
  - ▶ Στην ουσία μετατρέπει το αριθμό end-point σε αριθμό διεργασιών (θέση στον πίνακα διεργασιών)
  - ▶ Αν η διεργασία δεν εντοπιστεί – ή δεν είναι ζωντανή – επιστρέφει έναν αρνητικό αριθμό
- ▶ Χρησιμοποιεί την συνάρτηση virtual\_copy
  - ▶ ορίζεται στο αρχείο /usr/src/kernel/system.c
  - ▶ Μετατρέπει τις εικονικές διευθύνσεις σε πραγματικές με την χρήση της συνάρτησης umap\_local
  - ▶ Χρησιμοποιεί την συνάρτηση phys\_copy για την τελική ανηγραφή – επίπεδο assembly (klib386.s)



## Σύνοψη 12<sup>ης</sup> Διάλεξης

### Προηγούμενο Μάθημα

Ανταλλαγή Δεδομένων  
Ανιγραφή Μνήμης – Κλήσεις Πυρήνα

### Λειτουργικό Σύστημα Minix

Λειτουργίες Συστήματος Αρχείων  
Δομή Συστήματος Αρχείων  
Ανάγνωση Αρχείων

### Εύνοψη Μαθήματος

Εύνοψη Μαθήματος  
Βιβλιογραφία  
Επόμενη Διάλεξη



## 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



## Λίστα Μηνυμάτων (1)

Message	Input parameters	Reply Value
access	File name, access mode	Status
chdir	Name of new working directory	Status
chmod	File name, new mode	Status
chown	File name, new owner, group	Status
chroot	Name of new root directory	Status
close	File descriptor of file to close	Status
creat	Name of file to be created, mode	File descriptor
dup	File descriptor (for dup2, two fds)	New file descriptor
fcntl	File descriptor, function code, arg	Depends on function
fstat	Name of file, buffer	Status
ioctl	File descriptor, function code, arg	Status



## Λίστα Μηνυμάτων (2)

Message	Input parameters	Reply Value
link	Name of file to link to, name of link	Status
lseek	File descriptor, offset, whence	New position
mknod	File name, mode	Status
mknod	Name of dir or special, mode, address	Status
mount	Special file, where to mount, ro flag	Status
open	Name of file to open, r/w flag	File descriptor
pipe	Pointer to 2 file descriptors (modified)	Status
read	File descriptor, buffer, how many bytes	# Bytes read
rename	File name, file name	Status



## Λίστα Μηνυμάτων (3)

Message	Input parameters	Reply Value
rmdir	File name	Status
stat	File name, status buffer	Status
stime	Pointer to current time	Status
sync	(None)	Always OK
time	Pointer to place where current time goes	Status
times	Pointer to buffer for process and child times	Status
umask	Complement of mode mask	Always OK
umount	Name of special file to unmount	Status
unlink	Name of file to unlink	Status
utime	File name, file times	Always OK
write	File descriptor, buffer, how many bytes	# Bytes written



## Λίστα Μηνυμάτων (4)

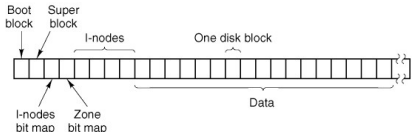
Message	Input parameters	Reply Value
exec	Pid	Status
exit	Pid	Status
fork	Parent pid, child pid	Status
setgid	Pid, real and effective gid	Status
setsid	Pid	Status
setuid	Pid, real and effective uid	Status
revive	Process to revive	(No reply)
unpause	Process to check	...

► Αυτά τα μηνύματα προέρχονται από τον διαχειριστή διεργασιών



## 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



## To 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 διαιρείτε 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 <sub>2</sub> (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



## Δομή super\_block (1)

- ▶ Ορίζεται στο αρχείο `/usr/src/servers/fs/super.h`
- ▶ Κατά την εκκίνηση του συστήματος φορτώνεται στην μνήμη

```
EXTERN struct super_block {
    /* # usable inodes on the minor device */
    ino_t s_ninodes;
    /* total device size, including bit maps etc */
    zonel_t s_nzones;
    /* # of blocks used by inode bit map */
    short s_imap_blocks;
    /* # of blocks used by zone bit map */
    short s_zmap_blocks;
    /* number of first data zone */
    zonel_t s_firstdatazone;
```



## Δομή super\_block (2)

```
/* log2 of blocks/zone */
short s_log_zone_size;
/* try to avoid compiler-dependent padding */
short s_pad;
/* maximum file size on this device */
off_t s_max_size;
/* number of zones (replaces s_nzones in V2) */
zone_t s_nzones;
/* magic number to recognize super-blocks */
short s_magic;
/* try to avoid compiler-dependent padding */
short s_pad2;
/* block size in bytes. */
unsigned short s_block_size;
```



## Δομή super\_block (3)

```
/* filesystem format sub-version */
char s_disk_version;

} super_block[NR_SUPERS];
```

- ▶ Ορίζεται ως πίνακας από NR\_SUPERS συσκευές
- ▶ ... το μέγιστο πλήθος συσκευών που μπορούμε να χειριζόμαστε ανά πάσα στιγμή



## Δομή super\_block (4)

- ▶ Για να επιταχυνθεί η απόδοση του συστήματος, οι υπολογισμοί του μεγέθους του inode bitmap, το inode που περιέχει τη ρίζα των φακέλων κλπ αποθηκεύονται στη δομή super\_block
- ▶ Αυτές όμως οι πληροφορίες δεν αποθηκεύονται στον αποθηκευτικό χώρο
- ▶ Υπάρχουν μόνο στη μνήμη

```
/* inode for root dir of mounted file sys */
struct inode *s_isup;
/* inode mounted on */
struct inode *s_imount;
/* precalculated from magic number */
unsigned s_inodes_per_block;
/* whose super block is this? */
dev_t s_dev;
```



## Δομή super\_block (5)

```
/* set to 1 iff file sys mounted read only */
int s_rd_only;
/* set to 1 iff not byte swapped file system */
int s_native;
/* file system version, zero means bad magic */
int s_version;
/* # direct zones in an inode */
int s_ndzones;
/* # indirect zones per indirect block */
int s_nindirs;
/* inodes below this bit number are in use */
bit_t s_isearch;
/* all zones below this bit number are in use */
bit_t s_zsearch;
```



## Αρχικοποίηση Συστήματος Αρχείων

- ▶ Η αρχικοποίηση ενός αποθηκευτικού χώρου γίνεται με την χρήση της mkfs

```
mkfs /dev/fd1 1440
```

- ▶ Δημιουργεί το superblock της συσκευής
- ▶ Το superblock διαβάζεται όταν καλούμε την mount (κλήση συστήματος)

- ▶ Βασίζεται στην συνάρτηση read\_super

```
_PROTOTYPE(int read_super, (struct super_block *s
```

- ▶ Προϋποθέτει ότι η συσκευή είναι ανοικτή - επικοινωνία με τον οδηγό της συσκευής



## Επικοινωνία με Οδηγό Συσκευών

- ▶ Η χρήση του αποθηκευτικού χώρου (συσσκευή Εισόδου/Εξόδου) γίνεται μέσω του οδηγού της συσκευής (device driver)
- ▶ Στο αρχείο `/usr/src/servers/fs/driver.c` ορίζονται όλες οι συναρτήσεις για την χρήση της συσκευής
  1. `dev_open` – FS opens a device
  2. `dev_close` – FS closes a device
  3. `dev_io` – FS does a read or write on a device
  4. `dev_status` – FS processes callback request alert
  5. `gen_orcl` – generic call to a task to perform an open/close
  6. `gen_io` – generic call to a task to perform an I/O operation
  7. `do_ioctl` – perform the IOCTL system call
  8. `do_setsid` – perform the SETSID system call (FS side)
  9. ...
- ▶ Οι συναρτήσεις αναλαμβάνουν να στείλουν μηνύματα στον κατάλληλο οδηγό της συσκευής (system call)



## 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



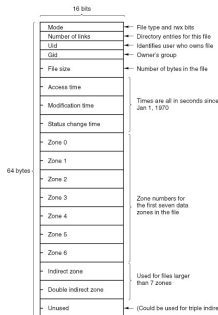
## Bitmaps -- Υπολογισμός Μεγέθους

- ▶ Με βάση την πληροφορία που διατηρεί το superblock μπορούμε να υπολογίσουμε το πλήθος των blocks
- ▶ Αναλόγως το πλήθος, υπολογίζουμε πόσα blocks απαιτούνται για να αποθηκεύουμε το bitmap
  - ▶ Αν το μέγεθος των block είναι 1KB -- χωράει ένα bitmap με 8192 εγγραφές
  - ▶ ... και για ένα bitmap από 10000 εγγραφές χρειαζόμαστε 2 blocks
- ▶ Zones
  - ▶ Το σύστημα αρχείων του MINIX 3 αποθηκεύει ζώνες blocks
  - ▶ ... για την αποθήκευση των blocks σε συνεχόμενα blocks
  - ▶ Στην πράξη δεν χρησιμοποιείται

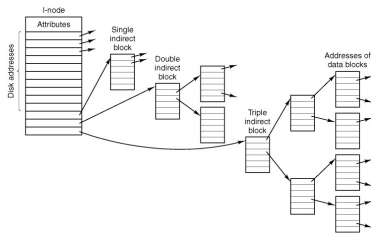


## Το i-node του MINIX 3

- ▶ 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



- ▶ 32-bit Zone Number (address)
- ▶ Block size = Zone size = 1KB
  - ▶ Με την χρήση των 7 διευθύνσεων zones – 7 KB πληροφορίας
  - ▶ Με την χρήση του indirect zone μπορούμε να αποθηκεύσουμε σε 1 block, 256 διευθύνσεις zones ( $\frac{1024 \times 8}{32}$ ) – 256 KB
  - ▶ Με την χρήση των second indirect zone μπορούμε να αποθηκεύσουμε σε 1 block, 256 διευθύνσεις από indirect zones, όπου κάθε indirect zone διατηρεί 256 διευθύνσεις zones – 64 MB
- ▶ Block size = Zone size = 4KB
  - ▶ Με την χρήση των 7 διευθύνσεων zones – 28 KB πληροφορίας
  - ▶ Με την χρήση του indirect zone μπορούμε να αποθηκεύσουμε σε 1 block, 1024 διευθύνσεις zones ( $\frac{4096 \times 8}{32}$ ) – 4 MB 1024\*
  - ▶ Με την χρήση των second indirect zone μπορούμε να αποθηκεύσουμε σε 1 block, 1024 διευθύνσεις από indirect zones, όπου κάθε indirect zone διατηρεί 1024 διευθύνσεις zones – 4 GB



- ▶ Τα triple indirect zone δεν υλοποιούνται στο MINIX 3

```

EXTERN struct inode {
    /* file type, protection, etc. */
    mode_t i_mode;
    /* how many links to this file */
    nlink_t i_nlinks;
    /* user id of the file's owner */
    uid_t i_uid;
    /* group number */
    gid_t i_gid;
    /* current file size in bytes */
    off_t i_size;

```

```

    /* time of last access (V2 only) */
    time_t i_atime;
    /* when was file data last changed */
    time_t i_mtime;
    /* when was inode itself changed (V2 only)*/
    time_t i_ctime;
    /* zone numbers for direct, ind, and dbl ind */
    zone_t i_zone[V2_NR_TZONES];
} inode[NR_INODES];

```

## Δομή i-node (3)

- ▶ Για να επαυχυθεί η απόδοση του συστήματος, ορισμένες πληροφορίες διατηρούνται μόνο στη μνήμη
- ▶ Δεν αποθηκεύονται στον αποθηκευτικό χώρο – υπάρχουν μόνο στη μνήμη

```
/* which device is the inode on */
dev_t i_dev;
/* inode number on its (minor) device */
ino_t i_num;
/* # times inode used; 0 means slot is free */
int i_count;
/* # direct zones (Vx_Nr_DZONES) */
int i_ndzones;
/* # indirect zones per indirect block */
int i_nindirs;
```



## Δομή i-node (4)

```
/* pointer to super block for inode's device */
struct super_block *i_sp;
/* CLEAN or DIRTY */
char i_dirt;
/* set to I_PIPE if pipe */
char i_pipe;
/* this bit is set if file mounted on */
char i_mount;
/* set on LSEEK, cleared on READ/WRITE */
char i_seek;
/* the ATIME, CTIME, and MTIME bits are here */
char i_update;
```



## Block Cache – Προσωρινή Αποθήκευση

- ▶ Στα μοντέρνα συστήματα οι συσκευές E/E είναι οι πιο 'αργές' σε σύγκριση με τον επεξεργαστή / μνήμη
- ▶ Η προσωρινή αποθήκευση των blocks στην μνήμη στοχεύει στην επιτάχυνση της ανάγνωσης δεδομένων από τις συσκευές E/E
- ▶ Σε μια κλήση συστήματος read (για την ανάγνωση δεδομένων από μια συσκευή E/E)
  - ▶ Ελέγχουμε αν τα block είναι αποθηκευμένα στην προσωρινή μνήμη (cache)
  - ▶ Όταν ένα block δεν υπάρχει στην cache ανατρέχουμε στην συσκευή E/E και τοποθετούμε στην cache
  - ▶ Πως αδειάζει η cache ?
- ▶ Βασίζονται σε κάποιο αλγόριθμο διαχείρισης προσωρινής αποθήκευσης – στρατηγική caching

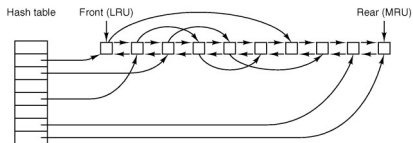


## Λειτουργία Block Cache στο MINIX 3

- ▶ Οι δίσκοι μεγαλώνουν σε μέγεθος → αυξάνεται το πλήθος των blocks
- ▶ Η μνήμη μεγαλώνει σε μέγεθος → αυξάνεται το μέγεθος της προσωρινής μνήμης (cache)
- ▶ Πρέπει να μπορούμε γρήγορα να εντοπίσουμε αν ένα block βρίσκεται στην cache
  - ▶ Ο έλεγχος γίνεται σε κάθε read (κλήση συστήματος)
- ▶ Χρησιμοποιούμε μια hash function
  - ▶ Βασίζεται στα τελευταία n bits της διεύθυνσης ενός block
  - ▶ Η συνάρτηση δεν είναι 1:1 – για περισσότερα του ενός blocks επιστρέφει την ίδια τιμή (hash value)
- ▶ Όλα τα blocks με την ίδια τιμή hash αποθηκεύονται μαζί σε μια συνδεδεμένη λίστα (linked list)



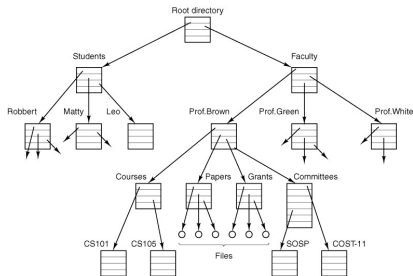
## Δομή Block Cache στο MINIX 3



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



## Φάκελοι και Μονοπάτια



## Διαχείριση Ανοικτών Αρχείων -- File Descriptors

- ▶ Για να χρησιμοποιήσουμε ένα αρχείο πρέπει πρώτα να το 'ανοίξουμε'
  - ▶ Να προετοιμάσουμε το σύστημα αρχείων με την κλήση open
- ▶ Στο τμήμα του πίνακα των διεργασιών που διατηρεί το σύστημα αρχείων αποθηκεύουμε τα αρχεία που έχει ανοίξει μια διεργασία

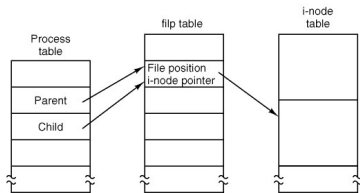
```
EXTERN struct fproc {  
    ...  
    struct inode *fp_workdir; /* pointer to working direct  
    struct inode *fp_rootdir; /* pointer to current root d  
    struct filp *fp_filp[OPEN_MAX]; /* the file descriptor  
    fd_set fp_filp_inuse; /* which fd's are in use? */  
    ...  
} fproc[NR_PROCS];
```



## Τρέχουσα Θέση Ανοικτού Αρχείου -- File Position

- ▶ Πρέπει να αποθηκεύουμε την 'τρέχουσα θέση' σε ένα ανοικτό αρχείο
  - ▶ Σε ποιο σημείο 'σταμάτησε' η τελευταία κλήση read
  - ▶ Χρησιμοποιούμε την κλήση lseek για να αλλάξουμε την θέση
- ▶ Στο τμήμα του πίνακα των διεργασιών που διατηρεί το σύστημα αρχείων αποθηκεύουμε τα αρχεία που έχει ανοίξει μια διεργασία
- ▶ Πού αποθηκεύουμε την τρέχουσα θέση;
  - ▶ Στον πίνακα των διεργασιών υπάρχει πρόβλημα όταν καλέσουμε τη fork
    - ▶ Θα αντιγραφούν τα ανοικτά αρχεία - σωστό
    - ▶ Θα αντιγραφούν οι θέσεις - λάθος
- ▶ Στο l-node του αρχείου
  - ▶ ... χειρότερα





- ▶ 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

## Βοηθητική συνάρτηση read (2)

```

PUBLIC ssize_t read(fd, buffer, nbytes)
int fd;
void *buffer;
size_t nbytes;
{
    message m;

    m.ml_i1 = fd;
    m.ml_i2 = nbytes;
    m.ml_p1 = (char *) buffer;
    return(_syscall(FS, READ, &m));
}
    
```

## Σύνοψη 12<sup>ης</sup> Διάλεξης

### Προηγούμενο Μάθημα

Ανταλλαγή Δεδομένων  
Αντιγραφή Μνήμης – Κλήσεις Πυρήνα

### Λειτουργικό Σύστημα Minix

Λειτουργίες Συστήματος Αρχείων  
Δομή Συστήματος Αρχείων  
Ανάγνωση Αρχείων

### Σύνοψη Μαθήματος

Σύνοψη Μαθήματος  
Βιβλιογραφία  
Επόμενη Διάλεξη

- ▶ Σύστημα Αρχείων στο Λ.Σ. MINIX 3
- ▶ Εσωτερική δομή συστήματος αρχείων στο Λ.Σ. MINIX 3
- ▶ Παράδειγμα Κλήσης Συστήματος Read

- ▶ 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