Διάλεξη 12:
Προχωρημένη Είσοδος/Έξοδος Χαμηλού Επιπέδου (Advanced Low-Level I/O) Κεφάλαιο 4 Stevens & Rago

Δημήτρης Ζεϊναλιπούρ

Περιεχόμενο Διάλεξης

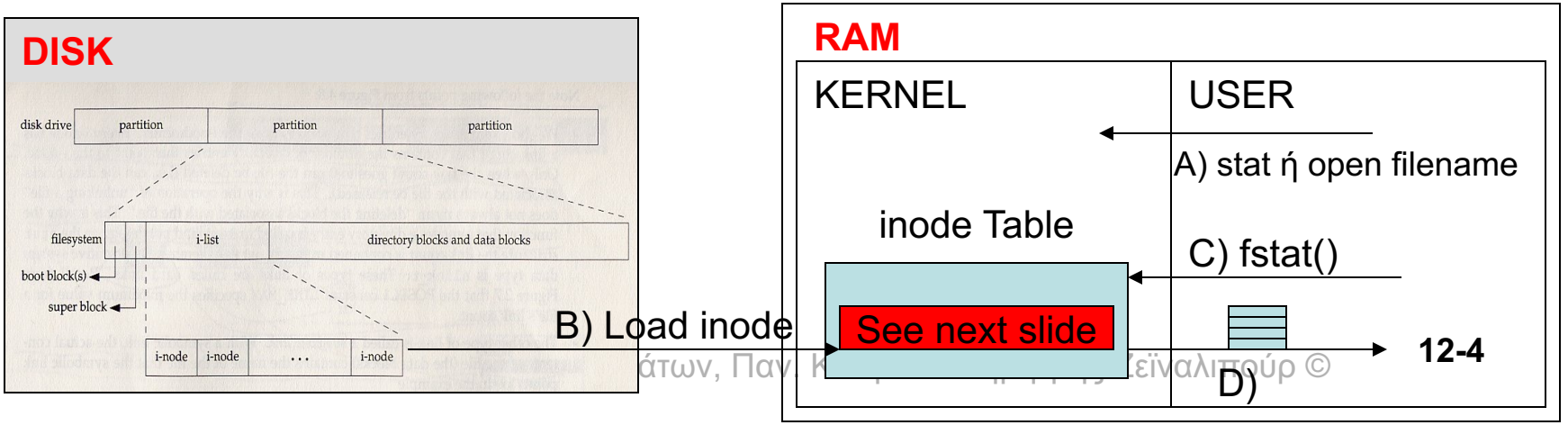
- Στην προηγούμενη διάλεξη μελετήσαμε τις εξής βασικές κλήσεις συστήματος για διαχείριση αρχείων (open, creat, read, write, lseek, close)
- Σε αυτή την ενότητα θα μελετήσουμε επιπλέον δυνατότητες του υπό-συστήματος αρχείων του πυρήνα.
- **Συγκεκριμένα θα μελετήσουμε**
  A. Διαχείριση Μέτα-πληροφορίων Αρχείων (sys/stat.h)
  B. Διαχείριση Αρχείων
  C. Διαχείριση Καταλόγων (dirent.h)
  D. Παραδείγματα Χρήσης

Μέτα-πληροφορίες Αρχείων

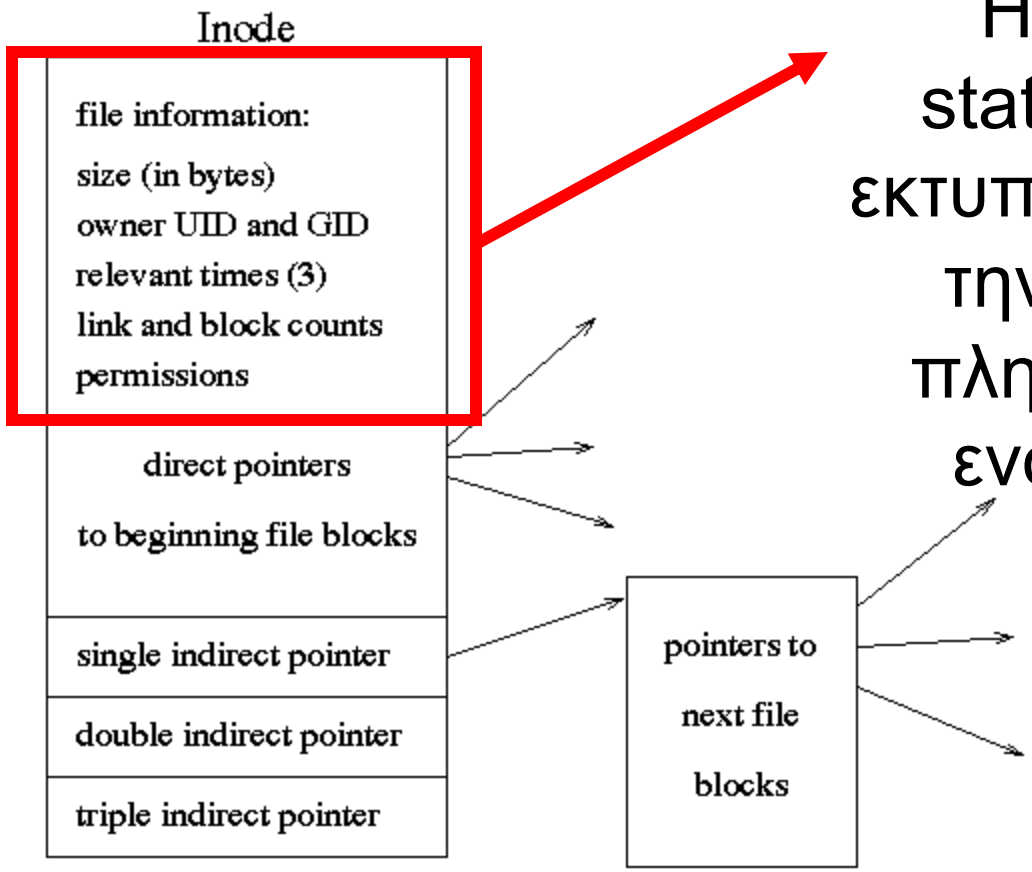
- Το open, creat, read, write, lseek, close μας επιτρέπει να έχουμε πρόσβαση στο **περιεχόμενο αρχείων.**
- **Τι γίνεται με τις υπόλοιπες πληροφορίες;** (όπως π.χ., αυτές που επιστρέφονται από την ls –al)…. δηλαδή filesize, permissions, last modification date, owner, κτλ.)
- Αυτές οι πληροφορίες ονομάζονται Μέτα-Δεδομένα (Meta-data) ή εναλλακτικά Μέτα-Πληροφορίες (Meta-information).
- Εδώ θα μελετήσουμε που αποθηκεύονται και πως ανακτώνται.

Μέτα-πληροφορίες Αρχείων
Που αποθηκεύονται?

- Γνωρίζουμε ότι το **inode (index node)** είναι μια δομή δευτερεύουσας μνήμης η οποία φορτώνεται στην κύρια μνήμη από τον πυρήνα όταν ανοίγει ένα αρχείο και η οποία περιέχει δείκτες στα πραγματικά δεδομένα.
- Στην Μνήμη υπάρχει ένα **I-node Table** το οποίο περιέχει τις μέτα-πληροφορίες των ανοικτών αρχείων.
- Πρόσβαση σε αυτές τις μέτα-πληροφορίες έχουμε μέσω των εντολών συστήματος **stat/fstat/lstat.**



Μέτα-πληροφορίες Αρχείων
Που αποθηκεύονται?



Η εντολές stat/fstat/lstat εκτυπώνουν αυτή την περιοχή πληροφοριών ενός i-node

Τα υπόλοιπα έχουν σχέση με την ανάκτηση των blocks που περιέχουν την πραγματική πληροφορία του αρχείου.

Μέτα-πληροφορίες Αρχείων
Το system call Stat()

- Για πρόσβαση στις μέτα-πληροφορίες εκτελούμε την κλήση συστήματος
- **#include <sys/stat.h>**

  int stat(char *path, struct stat *buf)

  Returns: -1=Error, 0=Success

  η οποία συμπληρώνει τα πεδία της δομής **buf** με τις πληροφορίες που είναι καταχωρημένες στο **i-node** του κόμβου με το όνομα path
- Εάν έχουμε ήδη ανοίξει το αρχείο τότε χρησιμοποιούμε τον file descriptor του ανοικτού αρχείου με την εντολή fstat.

  int fstat(int fd, struct stat *buf)
- Υπάρχει και η **lstat** η οποία θα μελετηθεί σε λίγο.

Μέτα-πληροφορίες Αρχείων
Εκτελώντας την stat μας επιστρέφετε…

…μεταξύ άλλων …. (δείτε το sys/stat.h για περισσότερα)

Τύπος και Πεδίο	Περιγραφή
ino_t **st_ino**	Αριθμός I-Node
nlink_t **st_nlink**	Αριθμός (σκληρών) συνδέσμων στο αρχείο (π.χ. ln oldfilename newfilename θα αυξήσει το nlink του oldfilename από 1 σε 2)
uid_t **st_uid**	UNIX userID (το ίδιο με αυτό στο /etc/passwd)
gid_t **st_gid**	UNIX groupID (το ίδιο με αυτό στο /etc/passwd)
off_t **st_size**	Μέγεθος Αρχείου σε bytes (εάν είναι regular αρχείο)
Τα πιο κάτω είναι το **πλήθος δευτερολέπτων** που έχουν παρέλθει από την 1/1/1970. Μπορούμε να τα μορφοποιήσουμε σε μια πιο εύληπτη μορφή με τις συναρτήσεις της βιβλιοθήκης time.h (συγκεκριμένα ctime) (δες παράδειγμα 1 πιο κάτω)	
time_t **st_atime**	time of last **a**ccess (of the **file's content**) – R
time_t **st_mtime**	time of last data **m**odification (of the **file's content**) – W or A
time_t **st_ctime**	time of status **c**hange (**inode change**) | ctime & mtime usually same
blksize_t **st_blksize**	Συνιστάμενο I/O Block για το αντικείμενο το οποίο μπορεί να διαφέρει μεταξύ συστημάτων αρχείων (π.χ., 4096 Bytes)
mode_t **st_mode**	Δικαιώματα Πρόσβασης Αρχείου (επόμενη διαφάνεια& παράδειγμα 2)

Μέτα-πληροφορίες Αρχείων

- *Το **stat.st_mode** μπορεί να αξιοποιηθεί με την χρήση των πιο κάτω macros (τα οποία ορίζονται μέσα στην sys/stat.h)*
  - ***S_ISLNK(st_mode) symbolic link***
  - ***S_ISREG(st_mode) regular file***
  - ***S_ISDIR(st_mode) directory***
  - ***S_ISCHR(st_mode) character device***
  - ***S_ISBLK(st_mode) block device***
  - ***S_ISFIFO(st_mode) fifo***
  - ***S_ISSOCK(st_mode) socket***
- *Η sys/stat.h περιέχει πολλές άλλες σταθερές τις οποίες καλείστε να μελετήσετε (man -s2 stat)*

Παράδειγμα 1: mystat

Να υλοποιήσετε σε C και με χρήση κλήσεων συστήματος, ένα απλό πρόγραμμα το οποίο να εκτυπώνει τις μέτα-πληροφορίες κάποιου αρχείου το οποίο δίδεται σαν παράμετρος.

π.χ.

./mystat /etc/passwd

Παράδειγμα 1: stat

Η εκτέλεση της εντολής stat του UNIX …

$ stat /etc/passwd

File: `/etc/passwd'

Size: 2005 Blocks: 8 IO Block: 4096 regular file

Device: fd00h/64768d Inode: 67570 Links: 1

Access: (0644/-rw-r--r--) Uid: (0/ root) Gid: (0/ root)

Access: 2009-01-13 13:35:33.000000000 +0200

Modify: 2008-03-11 12:46:59.000000000 +0200

Change: 2008-03-11 12:46:59.000000000 +0200

Μέγεθος Αρχείου σε bytes
du -b /etc/passwd

DeviceID (see man –s2 stat)

Συνιστάμενο Block Size

// Access Data

// Change Data

// Change Inode

ls -al file => counting in Bytes (/1000)
ls –alh file => counting in kB (/1024)

Παράδειγμα 1: mystat

```
#include <sys/stat.h>
#include <unistd.h> // STDOUT_FILENO
#include <stdio.h>  // printf()
#include <time.h>       // ctime()

int main(int argc, char *argv[]) {
      struct stat buf;
      printf("%s\n", argv[1]);
      if (stat(argv[1], &buf) < 0) {
            perror("lstat error");
            exit(1);
       }

       printf("+ I-Node: %li\n", buf.st_ino);
       printf("+ Size: %d\n", buf.st_size);
       printf("+ Hard Links: %d\n", buf.st_nlink);
       printf("+ User ID: %d\n", buf.st_uid);
       printf("+ Group ID: %d\n", buf.st_gid);
       printf("+ Last Content Access (atime): %s", ctime(&buf.st_atime));
       printf("+ Last I-Node Change (ctime): %s", ctime(&buf.st_ctime));
       printf("+ Last Content Change (mtime): %s", ctime(&buf.st_mtime));
       printf("+ Preferred I/O Block: %d\n", buf.st_blksize);
       printf("+ Allocated Blocks: %d\n", buf.st_blocks);
       return 0;
}
```

Παράδειγμα 1: Εκτέλεση mystat

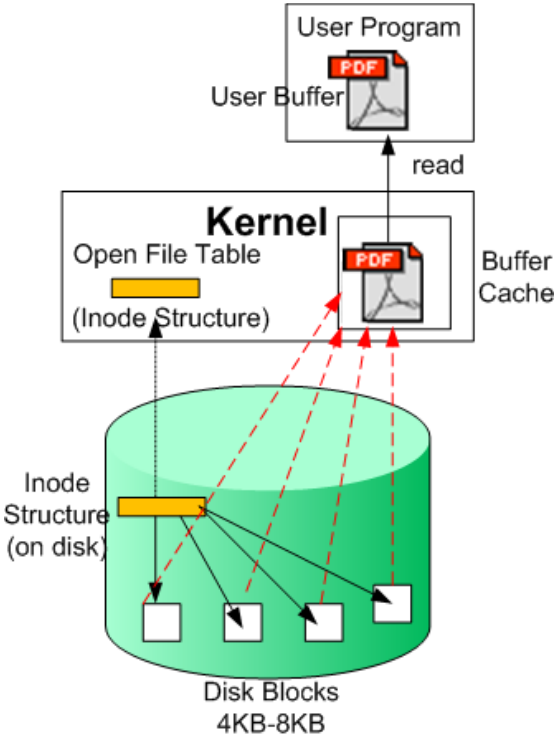
$./mystat /etc/passwd
/etc/passwd
+ I-Node: 67570
+ Size: 2005
+ Hard Links: 1
+ User ID: 0
+ Group ID: 0
+ Last Content Access (atime): Sat Jan 13 13:35:33 2009
+ Last I-Node Change (ctime): Tue Mar 11 12:46:59 2008
+ Last Content Change (mtime): Tue Mar 11 12:46:59 2008
+ Preferred I/O Block: 4096
+ Allocated Blocks: 8

Permissions, etc.

$ls -ial /etc/passwd
67570 -rw-r--r-- 1 root root 2005 Mar 11 2008 /etc/passwd

Flushing Data to Disk
fsync, fdatasync

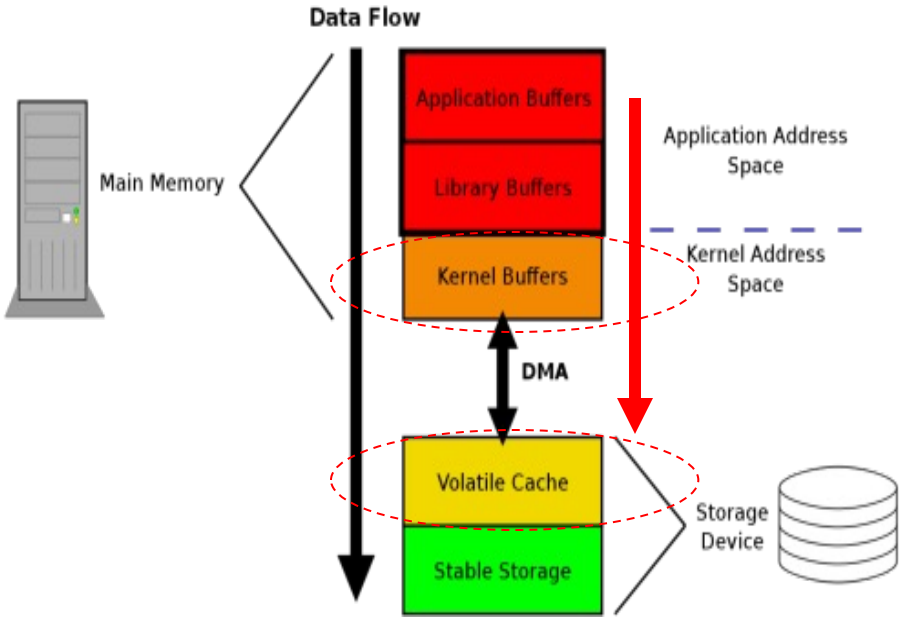
- Ανά πάσα στιγμή τα δεδομένα + μεταδομένα που έχουμε εγγράψει μέσω `write()` μπορεί να βρίσκονται στο **kernel space** αντί στο **δίσκο**.
- Για να βεβαιωθούμε ότι θα έχουν **κατεβεί σε επίπεδο controller μαγνητικού μέσου,** πρέπει να χρησιμοποιήσουμε τα system calls **`fsync, fdatasync(), κτλ.`**
  - Χρήσιμο σε εφαρμογές βάσεων δεδομένων όπου για λόγους συνέπειας στην εκτέλεση δοσοληψίων (transactions)



I/O Optimization I (Write-through Kernel)

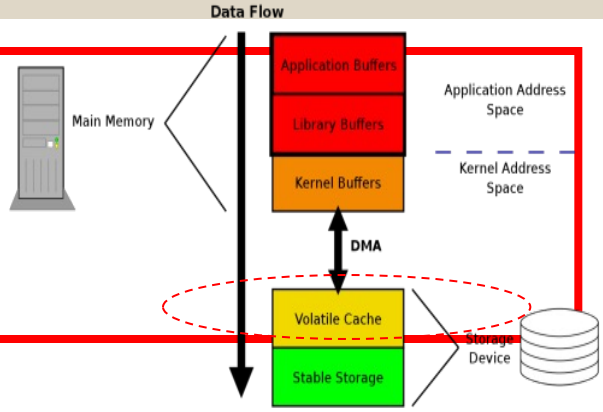
- **A) Kernel Buffers:** I/O operations performed against files opened with **O_DIRECT** bypass the kernel's page cache, writing directly to the storage.

- **B) Volatile Cache:** Recall that the storage may itself store the data in a write-back cache, so **fsync()** is still required for files opened with O_DIRECT in order to save the data to stable storage.



Flushing Data to Disk
fsync, fdatasync

```
#include <unistd.h>
int fsync(int fd);
int fdatasync(int fd);
```



- **fsync(): flushes metadata + data**
  - This includes **writing through** or flushing a **disk cache** if present.
  - The call will **block** until the device reports that transfer has completed

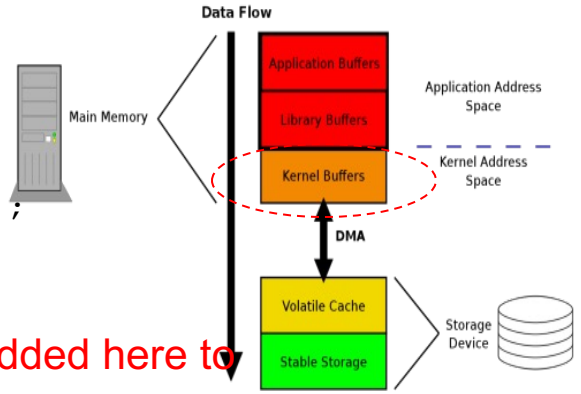
time_t	st_atime	time of last access (of the **file's content**) – R
time_t	st_mtime	time of last data modification (of the **file's content**) – W or A
time_t	st_ctime	time of status change (**inode change**) | ctime & mtime usually same

- **fdatasync(): flushes data only!**
  - "does not flush modified metadata **unless** that metadata is needed in order to allow a subsequent data retrieval to be correctly handled",
  - e.g., changes to *st_atime* or *st_mtime, as such, no metadata sync!*
- In Linux 2.2 and earlier, **fdatasync**() is equivalent to **fsync**(), and so has no performance advantage.

I/O Optimization I (Write-through Kernel)

```
#define _GNU_SOURCE
#include <string.h>
#include <stdlib.h>
#include <fcntl.h>
#define BLOCKSIZE 512

int main() {

        void *buffer;
        posix_memalign(&buffer, BLOCKSIZE, BLOCKSIZE);
        memcpy(buffer, image, sizeof(image));
        int f = open("feep.pgm",
                O_CREAT|O_TRUNC|O_WRONLY|O_DIRECT, S_IRWXU);
        write(f, buffer, BLOCKSIZE);
        close(f);
        free(buffer);
        return 0;
}
```

```
char image[] =
{
'P', '5', ' ', '2', '4', ' ', '7', ' ', '1', '5', '\n',
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 3, 3, 3, 3, 0, 0, 7, 7, 7, 7, 0, 0,11,11,11,11, 0, 0,15,15,15,15, 0,
0, 3, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0,11, 0, 0, 0, 0, 0,15, 0, 0,15, 0,
0, 3, 3, 3, 0, 0, 0, 7, 7, 7, 0, 0, 0,11,11,11, 0, 0, 0,15,15,15,15, 0,
0, 3, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0,11, 0, 0, 0, 0, 0,15, 0, 0, 0, 0,
0, 3, 0, 0, 0, 0, 0, 7, 7, 7, 7, 0, 0,11,11,11,11, 0, 0,15, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
};
```



fsync(f) or fdatasync(f) can be added here to overcome volatile cache of storage

posix_memalign() function shall allocate *size* bytes aligned on a boundary specified by *alignment*, and shall return a pointer to the allocated memory in *memptr*. (malloc on the other hand would align only to 8B, which is not what we want here (here we want 512B alignment)

Timestamps (Disabling atime)

- time updates are by far the biggest IO performance deficiency that Linux has today.
- To disable the writing of access times, you need to mount the filesystem(s) in question with the **noatime** option.

```
mount /home -o remount,noatime
```

- To make the change permanent, update your `/etc/fstab` and add `noatime` to the options field.

order in which filesystem checks are done at reboot time.

Before:

```
/dev/mapper/sys-home /home xfs defaults 0 2
```

After:

```
/dev/mapper/sys-home /home xfs nodev,nosuid,noatime 0 2
```

nodev - Don't interpret block special devices on the filesystem.

nosuid - Block the operation of suid, and sgid bits.

the **nomtime** mount option (on some unix) does *not* disable mtime updates, instead it performs a *lazy* mtime update - so the mtime is still updated, just delayed a little

I/O Optimization II (Disabling Atime)

- **O_NOATIME** (since Linux 2.6.8) Do not update the file last access time (*st_atime* in the inode) when the file is read(2). This flag can be employed only if one of the following conditions is true:
  - * The effective UID of the process matches the owner UID of the file.
  - * The calling process has the **CAP_FOWNER** capability in its user namespace and the owner UID of the file has a mapping in the namespace.
- This flag is intended for use by indexing or backup programs, where its use can significantly reduce the amount of disk activity.
  - This flag may not be effective on all filesystems. One example is NFS, where the server maintains the access time.
- More: http://man7.org/linux/man-pages/man2/open.2.html

Παράδειγμα 2: filetype

Να υλοποιήσετε σε C και με χρήση κλήσεων συστήματος, ένα απλό πρόγραμμα filetype το οποίο εκτυπώνει για κάθε αρχείο το οποίο δίδεται σαν παράμετρο, τον τύπο του αρχείου (regular, directory,…).

π.χ.,

*./filetype **

./filetype /etc/passwd /etc /dev/initctl /dev/log /dev/tty /dev/cdrom

Παράδειγμα 2: filetype

```
#include <sys/stat.h> // STAT related
#include <unistd.h> // STDOUT_FILENO

void printstat(struct stat *buf);

int main(int argc, char *argv[]) {
    struct stat buf;
    int i;

      for (i=1; i<argc; i++) {
         printf("%s:", argv[i]);
         if (lstat(argv[i], &buf) < 0) {
               perror("lstat error");
               continue;
         }

         printstat(&buf);
      }
      return 0;
}
```

```
void printstat(struct stat *buf) {
   char  *ptr;
   if (S_ISREG(buf->st_mode))
            ptr = "regular";
   else if S_ISDIR(buf->st_mode)
            ptr = "directory";
   else if S_ISCHR(buf->st_mode)
            ptr = "character special";
   else if S_ISBLK(buf->st_mode)
            ptr = "block special";
   else if S_ISFIFO(buf->st_mode)
            ptr = "fifo";
   else if S_ISLNK(buf->st_mode)
            ptr = "symbolic link";
   else if S_ISSOCK(buf->st_mode)
            ptr = "socket";
   else ptr = "Unknown Mode";
   printf("%s\n", ptr);
}
```

Γιατί lstat αντί stat? Σε περίπτωση symbolic link μας ενδιαφέρουν **τα metadata του ίδιου του link** (ότι είναι **symbolic link δηλαδή**) και όχι του αρχείου στο οποίο δείχνει το link … Περισσότερα στη συνέχεια…

Παράδειγμα 1: Εκτέλεση filetype

Αποτέλεσμα Εκτέλεσης

$ # Σημειώστε ότι το /dev/cdrom είναι symbolic link στο /dev/hda.

*$ls -al **/dev/cdrom***

*lrwxrwxrwx 1 root root 8 Feb 10 2003 **/dev/cdrom -> /dev/hda***

$./filetype /etc/passwd /etc /dev/initctl /dev/log /dev/tty /dev/cdrom /dev/hda

/etc/passwd:regular

/etc:directory

/dev/initctl:fifo

/dev/log:socket

/dev/tty:character special

/dev/cdrom:symbolic link

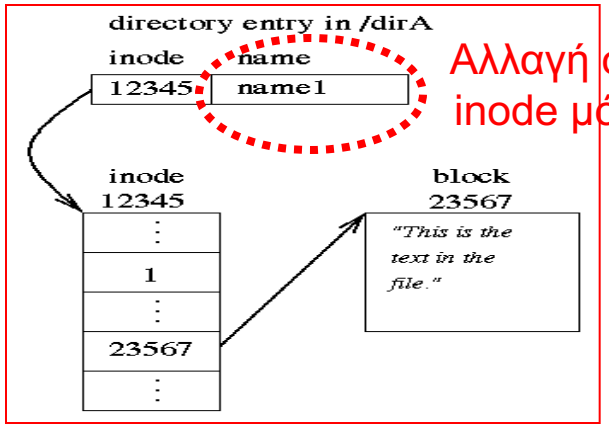
/dev/hda:block special

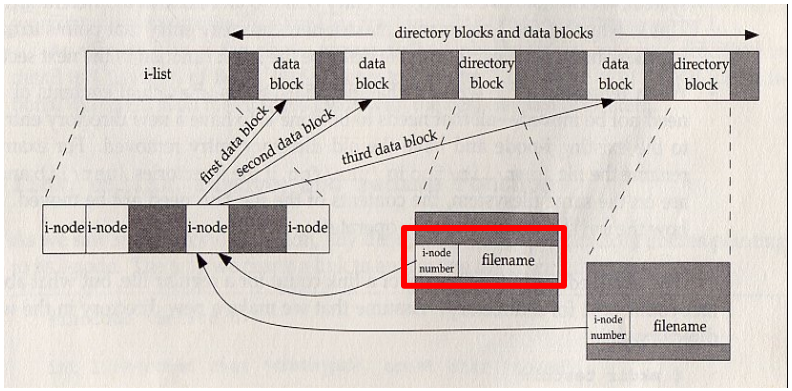
Η Κλήση Συστήματος rename()

int rename (char *oldpath, char *newpath)

Returns: -1=Error, 0=Success

- *Μετονομάζει τον κόμβο με το όνομα **oldpath** σε **newpath**.*
- *Ουσιαστικά η τροποποίηση γίνεται μέσα στο **directory block** το οποίο περιέχει το inode+όνομα του αρχείου*





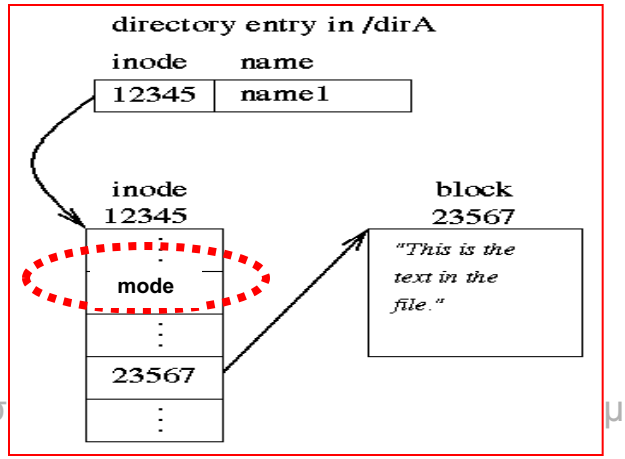
- *Το rename δουλεύει για οποιαδήποτε αρχεία (και καταλόγους) … είτε δίδονται με σχετική ή με απόλυτη διεύθυνση.*

Η Κλήση Συστήματος chmod()

int chmod (char *path, int mode)
Returns: -1=Error, 0=Success

Manipulate File Descriptor

- *Αλλάζει τα δικαιώματα προστασίας του κόμβου με όνομα path σε αυτά που περιγράφονται από το mode κατά τον γνωστό τρόπο (σταθ.* **S_Ixxxx στο fcntl.h ή ακέραιο)**
- *Η αλλαγή γίνεται μέσα στο inode όπως φαίνεται πιο κάτω*
- *Υπάρχει και αντίστοιχη κλήση συστήματος* ***fchmod****, η οποία αντί για path* ***περιμένει*** *ένα file descriptor (περιγραφέα αρχείου)*

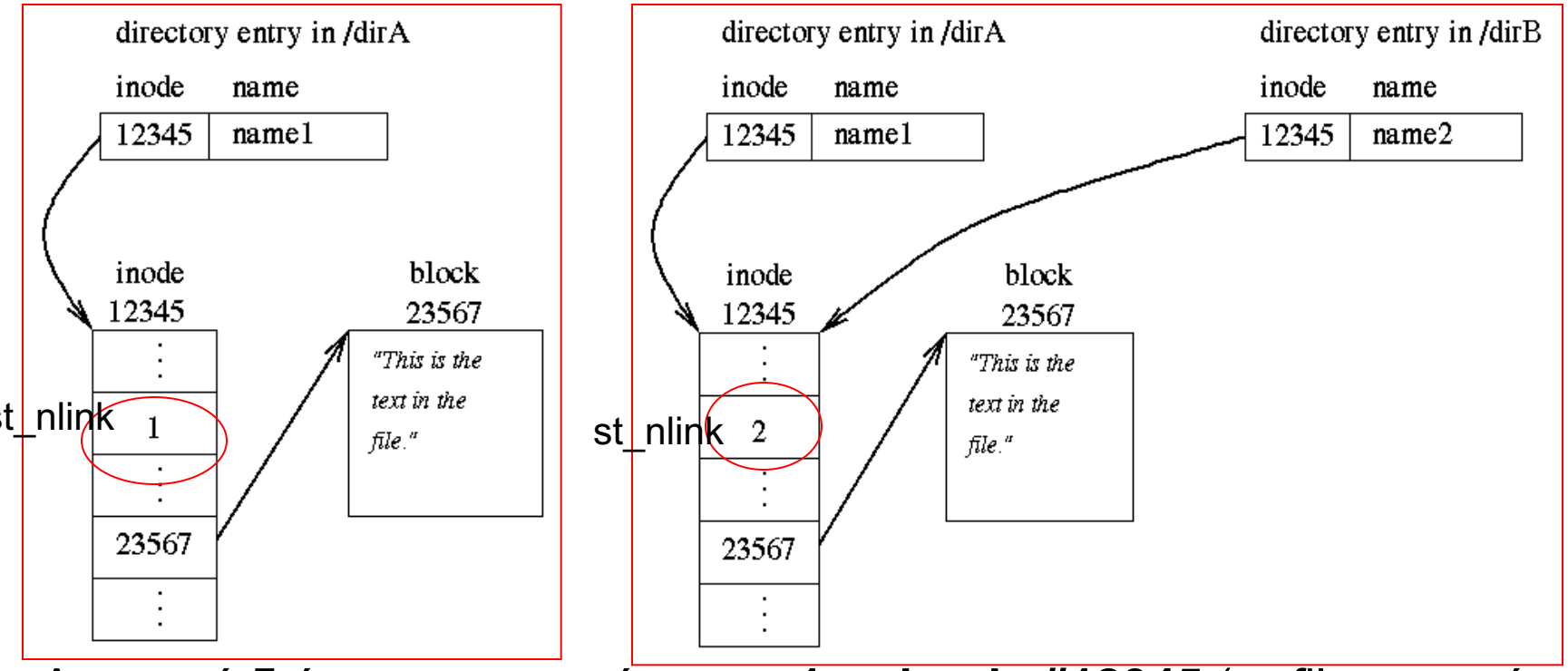


Οι Κλήσεις Συστήματος link() / unlink()

#include <unistd.h>
int link(char *oldpath, char *newpath)
int unlink(char *path)
Returns: -1=Error, 0=Success

- *Η **link** δημιουργεί ένα σκληρό σύνδεσμο **newpath** στο αρχείο **FILE** το οποίο έχει όνομα **oldpath**.*
- *Με αυτό τον τρόπο το **FILE**.**st_nlink*** (πεδίο του INODE) αυξάνεται κατά ένα (δες επόμενη διαφάνεια).
- *Η **unlink** διαγράφει τον σκληρό σύνδεσμο.*
- *Ουσιαστικά απλά μειώνεται κατά ένα ο μετρητής **FILE**.**st_nlink***
- *Εάν ο μετρητής γίνει ίσος με 0 τότε διαγράφονται τα blocks του αρχείου **FILE** από την δευτερεύουσα μνήμη.*

Οι Κλήση Συστήματος link()



- Αριστερά δείχνουμε ο αρχείο **name1** με **inode #12345** (το filename είναι εντελώς αχρείαστο πλέον!) το οποίο έχει **stat.st_nlink=1.**
- Δεξιά δείχνουμε την περίπτωση που έχει δημιουργηθεί ένα hard link μέσω της “ln name name2” ή μέσω της link(). Τώρα **το stat.st_nlink=2.**

Links Advanced

- Directories may not be hardlinked

```
$ ln a c
ln: a: Is a directory
```

- Hard links may not span file systems.

```
$ ln /tmp/a target.txt
ln: failed to create hard link ‘target.txt’ => ‘/tmp/a’: Invalid cross-device link
```

- Symbolic Links possible to Symbolic Links
  - LINK1 => LINK2 => FILE. FILE cannot l

```
$ ln -s /tmp/a target.txt # Symbolic Links possible even when on different file systems
$ ln -s target.txt newtarget.txt # Creating Symbolic to Symbolic
$ ls -ial
total 8
15036451919 drwx------  2 dzeina faculty   55 Oct 25 09:52 .
 6442451075 drwx-----x 49 dzeina faculty 4096 Oct 25 09:53 ..
15036451921 lrwxrwxrwx  1 dzeina faculty   10 Oct 25 09:52 newtarget.txt -> target.txt
15036451920 lrwxrwxrwx  1 dzeina faculty    6 Oct 25 09:52 target.txt -> /tmp/a
```



```
$ ls -ial /tmp/a
68 -rw------- 1 dzeina faculty 0 Oct 25 09:49 /tmp/a
```

Directories & Inodes

- Directories, like files have inodes but these do not contain any pointers to data blocks.

```
$ stat a
# the debugfs command is the more advanced command to play around with inodes but requires
root.
  File: ‘a’
  Size: 10        Blocks: 0          IO Block: 32768  directory
Device: 33h/51d Inode: 15036451922  Links: 2
Access: (0700/drwx------)  Uid: ( 7240/  dzeina)   Gid: ( 2531/ faculty)
Access: 2023-10-25 09:56:11.908492347 +0300
Modify: 2023-10-25 09:56:11.908492347 +0300
Change: 2023-10-25 09:56:11.908492347 +0300
 Birth: -
```

- Let’s see how blocks increase on a regular file

```
$ ls –s # print the allocated size of each file, in blocks
total 0
0 a  0 b  0 newtarget.txt  0 target.txt
$ echo “Hello World” > target.txt # writing to actual file
$ cat /tmp/a
Hello World
$ ls –s /tmp/a
4 /tmp/a
```

Οι Κλήσεις Συστήματος symlink() / readlink()

#include <unistd.h>
int symlink(char *oldpath, char *newpath)
int readlink(char *path, char *buf, int size)
Returns: -1=Error, 0=Success and readline returns the final number of bytes read to buf.

- *Η symlink δημιουργεί ένα συμβολικό σύνδεσμο από την oldpath στην newpath (όπως η ln –s oldpath newpath).*

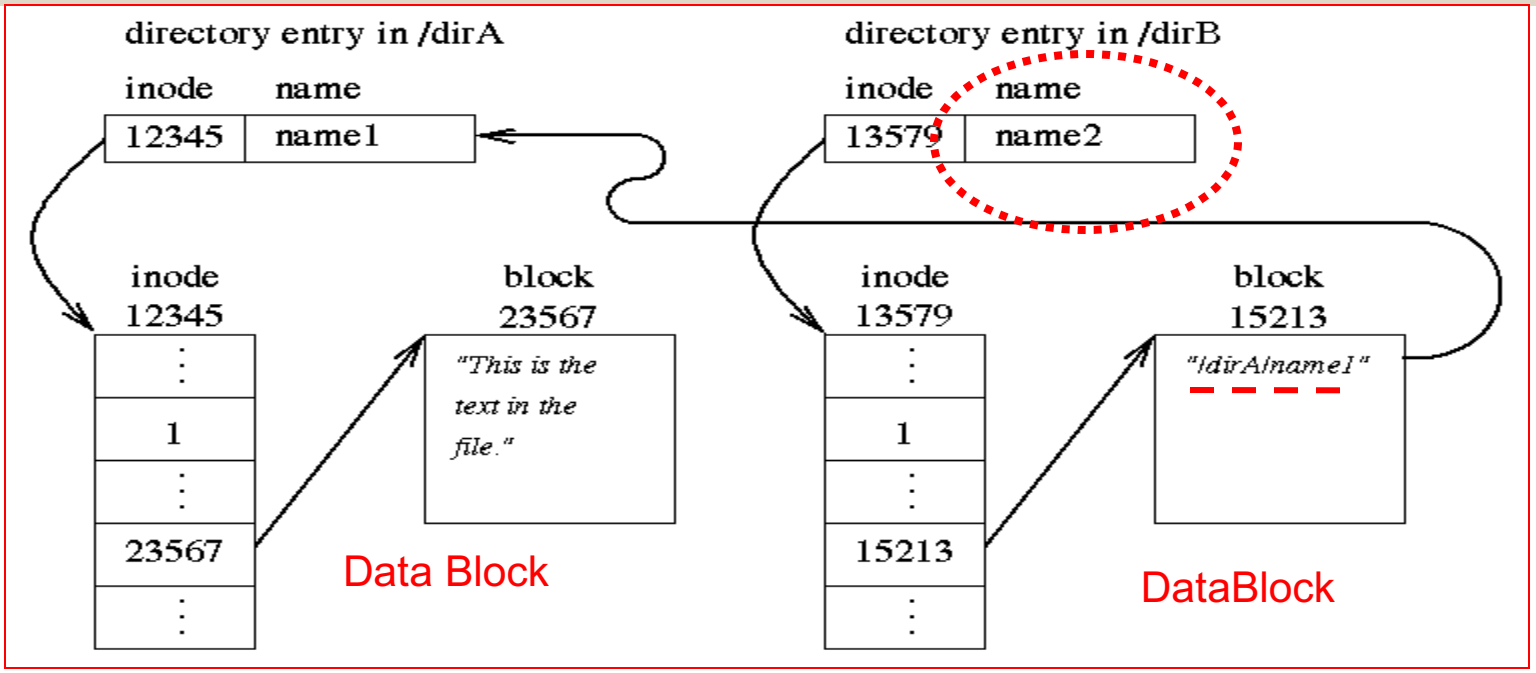
- *Η readlink επιστρέφει στο buf (μεγέθους size bytes), το όνομα στο οποίο δείχνει ο συμβολικός σύνδεσμος.* ***oldpath*** *-> newpath*
- *Η συνάρτηση* ***readlink*** *επιστρέφει σαν τιμή εξόδου τον αριθμό των bytes που διαβάστηκαν στο buf.*

π.χ. *char buffer[20]; int size = 0;*

```
symlink("/tmp/crawler", "mycrawler");
size = readlink("mycrawler", &buffer, 20);
buffer[size]='\0';
printf("%s %d\n", buffer, size);
```

=> Εκτυπώνει /tmp/crawler 12

Συμβολικοί Σύνδεσμοι, symlink και stat.st_nlink



- Δεξιά φαίνεται ότι το dirB/name2 είναι symbolic link στο /dirA/name1 (μέσω της εντολής ``ln –s /dirA/name1 dirB/name2'' ή symlink("/dirA/name1", dirB/name2)
- Επειδή το symbolic link δημιουργεί ένα νέο **inode#13579** με **stat.st_nlink=1**
- **Σημείωση:** Εάν θέλουμε τις πληροφορίες για το inode ενός symbolic link (και όχι του αρχείου το οποίο αναφέρεται από το link) δηλαδή του **inode#13579** αντί του **inode#12345**, τότε χρησιμοποιούμε:

  int lstat(char *path, struct stat *buf)

  Και ΟΧΙ την

  int stat(char *path, struct stat *buf)

Symbolic Loops

- Creating Loops with Symbolic Links possible, but not dangerous (just a set of orphan pointers) ☺

```
$ ls -al
total 0
drwx------ 2 dzeina faculty  46 Oct 25 10:27 .
drwx------ 5 dzeina faculty 120 Oct 25 10:26 ..
-rw------- 1 dzeina faculty   0 Oct 25 10:26 a
lrwxrwxrwx 1 dzeina faculty   1 Oct 25 10:27 b -> a
lrwxrwxrwx 1 dzeina faculty   1 Oct 25 10:27 c -> b
$ rm a  # delete the actual file
rm: remove regular empty file ‘a’? y
$ ls -al # show orphan symbolic links
total 0
drwx------ 2 dzeina faculty  34 Oct 25 10:27 .
drwx------ 5 dzeina faculty 120 Oct 25 10:26 ..
lrwxrwxrwx 1 dzeina faculty   1 Oct 25 10:27 b -> a
lrwxrwxrwx 1 dzeina faculty   1 Oct 25 10:27 c -> b

$ ln -s c a # substitute target with a symbolic link that creates a loop
$ ls -al
total 0
drwx------ 2 dzeina faculty  46 Oct 25 10:28 .
drwx------ 5 dzeina faculty 120 Oct 25 10:26 ..
lrwxrwxrwx 1 dzeina faculty   1 Oct 25 10:28 a -> c
lrwxrwxrwx 1 dzeina faculty   1 Oct 25 10:27 b -> a
lrwxrwxrwx 1 dzeina faculty   1 Oct 25 10:27 c -> b
```

Corrupting the file system (for fun and to dive deeper into the topic) possible by root using more advanced tools

https://www.linux.com/training-tutorials/fun-e2fsck-and-debugfs/

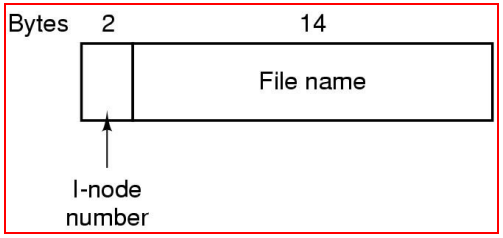
Διαχείριση Καταλόγων
Οι Κλήσεις Συστήματος mkdir () / rmdir()

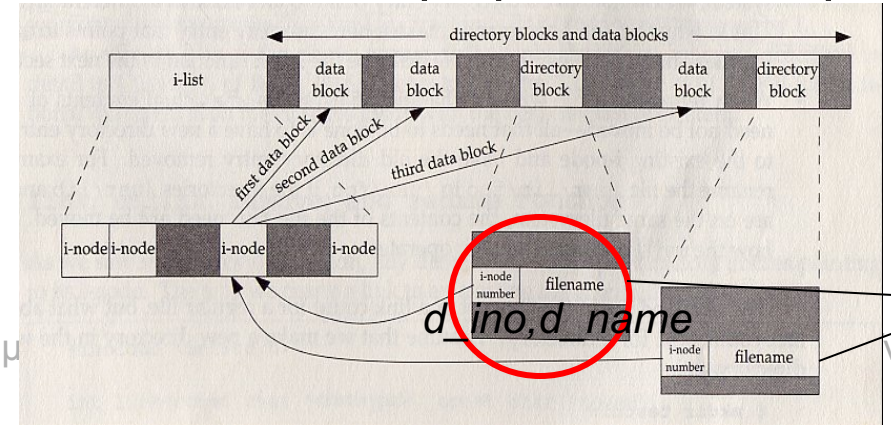
#include <sys/stat.h>
int mkdir (char *path, int mode)
int rmdir(char *path)
Returns: -1=Error, 0=Success

- *Η mkdir δημιουργεί ένα νέο κατάλογο με όνομα path και δικαιώματα προστασίας mode (π.χ., rwx------ = 700)*
- *Το path είναι σχετικό (π.χ., tmp) ή απόλυτο (π.χ., /tmp/f1)*
- *Σημειώστε ότι δικαιώματα τα οποία **δεν επιτρέπονται** από την τρέχουσα τιμή του umask δε δίνονται στον κατάλογο. Π.χ. umask 022 => 755 (δηλαδή το umask περιορίζει την εντολή αυτή)*
- *Επομένως εάν δώσουμε 777 τότε η umask θα θέσει τελικά τα permissions όπως τα θέλει.*
- *Η **rmdir** διαγραφεί τον κατάλογο με το όνομα path, εφόσον ο κατάλογος είναι κενός. **Αυτή η προϋπόθεση υπάρχει για να μην μένουν τα blocks αρχείων και τα inodes τους ορφανά!***
- *Ένα κοινό λάθος είναι να δώσουμε 600 (rw-) δικαιώματα. Οι κατάλογοι χρειάζονται τουλάχιστο (rwx) στο USER, δηλαδή 700 για παρουσίαση των αποτελεσμάτων της ls.*

Διαχείριση Καταλόγων
Προσπέλαση Καταλόγων

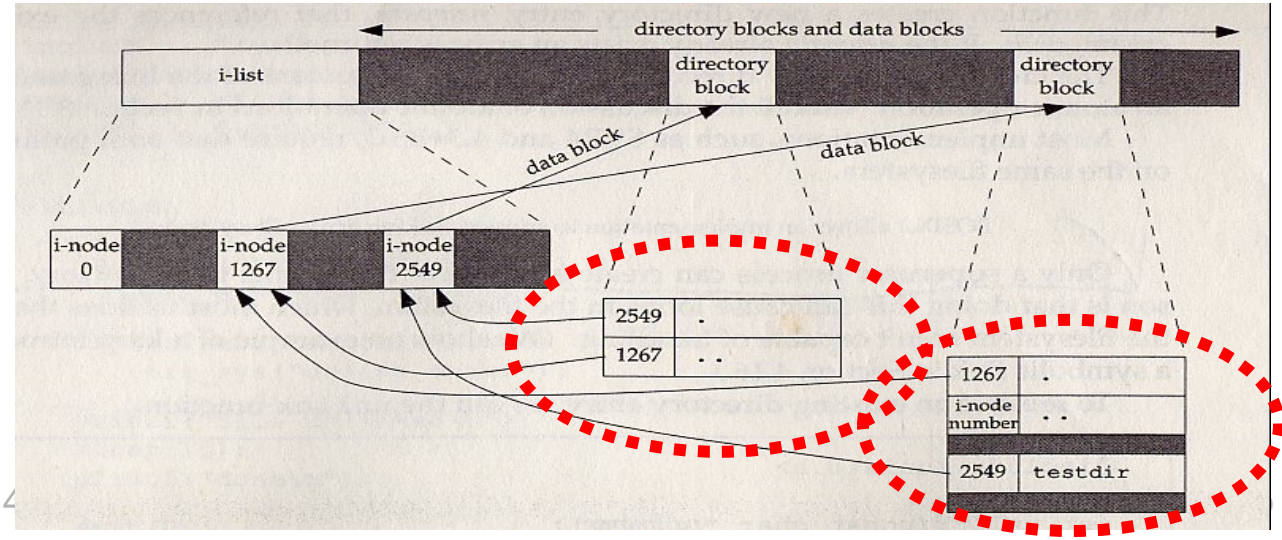
- *Τα περιεχόμενα καταλόγων τα οποία περιέχουν μια λίστα από (d_ino,d_name) μπορούν να προσπελασθούν μέσω των **συναρτήσεων βιβλιοθήκης** (όχι κλήσεις συστήματος) **opendir, readdir και closedir** (το d_name είναι συνήθως 255 chars)*
- *Η πρόσβαση σε ένα κατάλογο γίνεται μέσω ενός δείκτη **DIR *** (ανάλογου με τον FILE *) που χρησιμοποιείται στην συνάρτηση βιβλιοθήκης stdio.h*
- *Ωστόσο **μόνο ο πυρήνας μπορεί να γράψει** στο περιεχόμενο ενός καταλόγου (εν αντίθεση με τα κοινά αρχεία). Αυτό συμβαίνει για να προστατέψει ο πυρήνας τον χρήστη από λάθη τα οποία θα καταστρέψουν το δένδρο καταλόγων*





Διαχείριση Καταλόγων
Προσπέλαση Καταλόγων

- *Σημειώστε ότι κάθε αρχείο καταλόγου περιέχει στην αρχή του dir block τις i-node διευθύνσεις:*
  - *«.» υφιστάμενου καταλόγου και*
  - *«..» προηγουμένου καταλόγου*
- *Αυτό γίνεται για να είναι εφικτή η πλοήγηση προς τα πάνω στο κατάλογο του υποσυσ. αρχείων*



Γονικός κατάλογος

Προσπέλαση Καταλόγων

Οι Κλήσεις βιβλιοθήκης opendir (), closedir(), readdir()

#include <dirent.h>

DIR *opendir (char *path)

Returns: NULL=Error else pointer to DIR.

int closedir(DIR *dp)

Returns: -1=Error, 0=Success

```
struct dirent {
    ino_t       d_ino;      /* inode number */
    char        d_name[256]; /* filename */
};
```

- *Η opendir ανοίγει τον κατάλογο με όνομα path και επιστρέφει ένα δείκτη σε DIR για την πρόσβαση στον κατάλογο.*
- *Η closedir κλείνει τον κατάλογο το οποίο έχει ανοίξει μέσω του *dp*

#include <dirent.h>

struct dirent *readdir (DIR *dp)

- *Διαβάζει το επόμενο entry του ανοικτού καταλόγου dp.*
- *Επιστρέφει ένα δείκτη σε δομή struct dirent που αντιστοιχεί στο τρέχον στοιχείο του περιεχομένου του καταλόγου (από τον δείκτη dp)*
- *Επιστρέφει NULL όταν δεν υπάρχουν άλλα στοιχεία για διάβασμα.*

Διαχείριση Καταλόγων
Προσπέλαση Καταλόγων

#include <unistd.h>

int chdir(const char *path); και fchdir(int filedes)

Return: -1=Error, 0=Success

char *getcwd(char *buf, size_t size);

Return: NULL=Error, buf=Success

- Το chdir επιτρέπει σε ένα πρόγραμμα να αλλάξει τον τρέχων κατάλογο (όπως την εντολή cd).
- Ο τρέχον κατάλογος εδώ ορίζεται μέσα στις εσωτερικές δομές της διεργασίας και δεν αναφέρεται στο «.» του inode.
- Για να βρείτε τον **τρέχων κατάλογο** εκτελέστε την συνάρτηση συστήματος **getcwd** (δηλ., όμοιο με την "pwd")
- Η getcwd γράφει το όνομα του τρέχων καταλόγου στο buf (μεγέθους size) όπως η εντολή ***readlink που είδαμε προηγουμένως.***

Παράδειγμα 3: lsdirR

Να υλοποιήσετε σε C και με χρήση κλήσεων συστήματος, ένα απλό πρόγραμμα lsdirR(pathname) το οποίο εκτυπώνει αναδρομικά όλα τα αρχεία τα οποία αναφέρονται μέσω του pathname.

π.χ.

./lsdir ~/public_html/courses/epl111

Παράδειγμα 3: lsdirR

```
#include <stdio.h> // printf
#include <sys/types.h>
// opendir, readdir, closedir
#include <dirent.h>
#include <sys/stat.h> // lstat

// function prototype
void printdir(char *, int);

int main(int argc, char *argv[])
{
   printf("Directory scan of %s:\n",
     argv[1]);
   printdir(argv[1], 3);
   printf("done.\n");
   exit(0);
}
```

Κενά παραγράφου

```
void printdir(char *dir, int indent) {
      DIR *dp;
      struct dirent *entry;
      struct stat statbuf;
      if((dp = opendir(dir)) == NULL) {
        perror(dir);    return;
      }
      chdir(dir);          // change directory

      while((entry = readdir(dp)) != NULL) {
            lstat(entry->d_name,&statbuf);
            if(S_ISDIR(statbuf.st_mode)) {
                  /* Found a directory, but ignore . and .. */
                  if(strcmp(".",entry->d_name) == 0 ||
                    strcmp("..",entry->d_name) == 0)
                        continue;
                  printf("%*s%s/\n",indent,"",entry->d_name);
                  /* Recurse using a new indent offset */
                  printdir(entry->d_name,indent);
            }
            else printf("%*s%s\n",indent,"",entry->d_name);
      }
      chdir("..");
      closedir(dp);
}
```

Παράδειγμα 1: Εκτέλεση lsdirR

Αποτέλεσμα Εκτέλεσης $./ldirR ~/public_html/courses/epl111

Directory scan of /home/faculty/dzeina/public_html/courses/epl111:
contract.pdf
exercises/
ex1.pdf
ex2.pdf
exercises.html
index.html
lock.gif
notes.html
pdf.gif
proofwriting.pdf
rosen.png
slides/
lect3.pdf
lect1.pdf
…
lect15.pdf
ucy.gif
ex-manual.pdf
done.

Σημείωση: Δεν υπάρχει κάποια συγκεκριμένα σειρά

mmap: map or unmap files or devices into memory

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>

#define FILEPATH "/tmp/mmapped.bin"
#define NUMINTS  (1000)
#define FILESIZE (NUMINTS * sizeof(int))

/* http://en.wikipedia.org/wiki/Memory-mapped_file */
int main(int argc, char *argv[]) {
    int i;
    int fd;
    int result;
    int *map;  /* mmapped array of int's */

    /* Open a file for writing.
     *  - Creating the file if it doesn't exist.
     *  - Truncating it to 0 size if it already exists. (not really needed)
     *
     * Note: "O_WRONLY" mode is not sufficient when mmaping.
     */
    fd = open(FILEPATH, O_RDWR | O_CREAT | O_TRUNC, (mode_t)0600);
    if (fd == -1) {
        perror("Error opening file for writing");
        exit(EXIT_FAILURE);
    }
```

***mmap**() creates a new mapping in the virtual address space of the calling process. The starting address for the new mapping is specified in addr. The length argument specifies the length of the mapping (which must be greater than 0).*

mmap: map or unmap files or devices into memory

```
/* Stretch the file size to the size of the (mmapped) array of ints
 */
result = lseek(fd, FILESIZE-1, SEEK_SET);
if (result == -1) {
    close(fd);
    perror("Error calling lseek() to 'stretch' the file");
    exit(EXIT_FAILURE);
}


/* Something needs to be written at the end of the file to have the file actually have the new size */
result = write(fd, "", 1);
if (result != 1) {
    close(fd);
    perror("Error writing last byte of the file");
    exit(EXIT_FAILURE);
}
```

The *prot* argument describes the desired memory protection of the mapping (and must not conflict with the open mode of the file). It is either **PROT_NONE** or the bitwise OR of one or more of the following flags: **PROT_EXEC** Pages may be executed. **PROT_READ** Pages may be read. **PROT_WRITE** Pages may be written. **PROT_NONE** Pages may not be accessed.

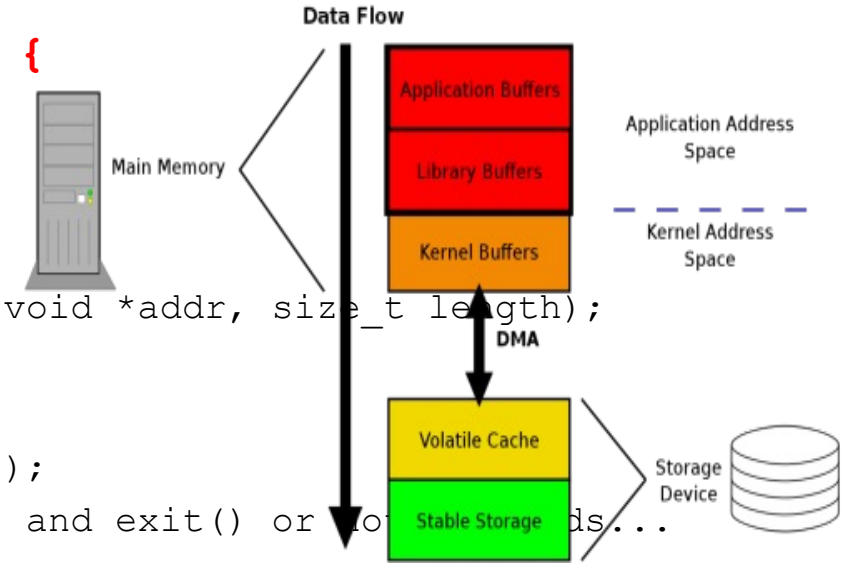
MAP_SHARED Share this mapping. Updates to the mapping are visible to other processes mapping the same region (child, parent, sibiling)

```
/* Now the file is ready to be mmapped. */
map = mmap(0, FILESIZE, PROT_READ | PROT_WRITE,
                                    MAP_SHARED, fd, 0);

if (map == MAP_FAILED) {
    close(fd);
    perror("Error mmapping the file");
    exit(EXIT_FAILURE);
}
```

void *mmap(void *_addr_, size_t _length_, int _prot_, int _flags_, int _fd_, off_t _offset_);

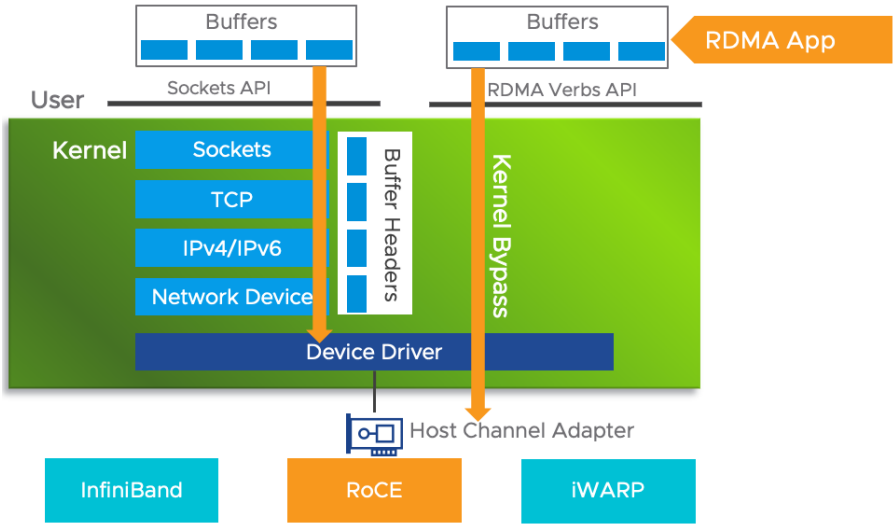
mmap: map or unmap files or devices into memory

```
/* Now write int's to the file as if it were memory (an array of ints).
     */
     for (i = 1; i <=NUMINTS; ++i) {
          map[i] = 2 * i;
     }

    /* free the mmapped memory: int munmap(void *addr, size_t length);
     */
    if (munmap(map, FILESIZE) == -1) {
        perror("Error un-mmapping the file");
        /* Decide here whether to close(fd) and exit() or not. Depends...
*/
    }

    /* Un-mmaping doesn't close the file, so we still need to do that.
     */
    close(fd);
    return 0;
}
```



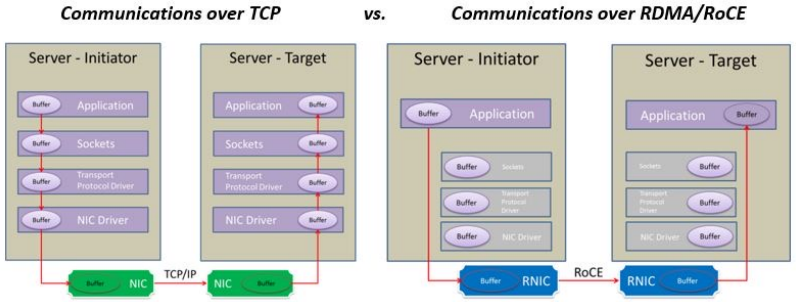
RDMA

- In computing, **remote direct memory access** (**RDMA**) is a direct memory access from the memory of one computer into that of another without involving either one's operating system. This permits high-throughput, low-latency networking, which is especially useful in massively parallel computer clusters.



RDMA

- Advantages
  - Lower Latency
  - Lower CPU, caching and context switching
- Disadvantage:
  - the target node is not notified of the completion of the request (single-sided communications).
    - TCP ACKs provide a natural mechanism to cope with producer-consumer data rates
  - **RDMA operates over an inherently reliable link layer whereas TCP typically operates over the unreliable Ethernet link layer**.



RDMA over Converged Ethernet or Infinband

RDMA over Converged Ethernet (RoCE) or InfiniBand over Ethernet (IBoE)

- **RoCE** defines how to perform RDMA over Ethernet
  - The RoCE v2 protocol exists on top of either the UDP/IPv4 or the UDP/IPv6 protocol
    - UDP destination port number 4791 has been reserved for RoCE v2
    - Installation Example: https://support.hpe.com/hpesc/public/docDisplay?docId=a00071081en_us&docLocale=en_US&page=GUID-617F4C95-AA58-43F7-B524-78C6535747AC.html
  - the iWARP protocol defines how to perform RDMA over a connection-oriented transport like the Transmission Control Protocol (TCP).
- **InfiniBand** architecture specification defines how to perform RDMA over an InfiniBand network.
  - **InfiniBand (IB) is a network specially designed for RDMA**, which guarantees reliable transmission at the hardware level, but