ATHENS UNIVERSITY
OF ECONOMICS
AND BUSINESS

Λειτουργικά Συστήματα

Συστήματα Αρχείων (File Systems)

Με μια ματιά…

- **File** (**Αρχείο**)
  - Περιλαμβάνει ένα σύνολο λογικά συσχετιζόμενων δεδομένων μη-προσωρινής φύσης (συνήθως αποθηκευμένο σε δίσκους).

- **File System** (**Σύστημα Αρχείων**): Μέρος ενός Λ.Σ. υπεύθυνο για την
  - **Naming** (Ονομασία)
  - **Organization** (Οργάνωση)
  - **Storage** (Αποθήκευση)
  - **Access** (Προσπέλαση)
  - **Protection** (Προστασία)
  - **Sharing** (Διαμοιρασμό)

    …των αρχείων

- Προσφέρει το **file abstraction**
  - Κρύβει όλες τις λεπτομέρειες διαχείρισης και προσπέλασης των συσκευών αποθήκευσης (π.χ., μαγνητικών δίσκων), οργάνωσης δεδομένων ενός αρχείου, κ.τ.λ.

Με μια ματιά…

- **Directory** (**Κατάλογος**)
  - Επιτρέπει ονόματα υψηλού επιπέδου (π.χ., ASCII strings) να ανατεθούν σε αυτό το **storage abstraction**.
  - π.χ. **/usr/bin/echo** ➔ σε κάποιο εσωτερικό **file id** (συνήθως ένας μεγάλος αριθμός που ορίζει ένα αρχείο στο σύστημα).

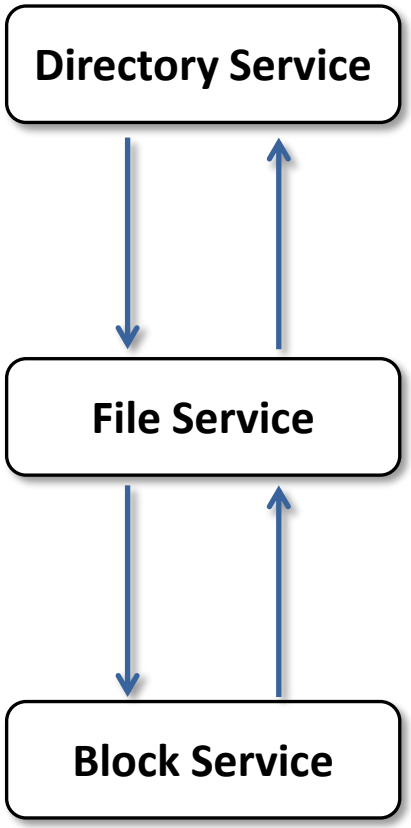
- Εξυπηρετεί τα:
  - **File naming** (**ονομασία**)
    - Η μετάφραση ενός filename στο αντίστοιχο file ID.
  - **Access Control** (**Έλεγχος πρόσβασης**)
    - Περιορίζει την πρόσβαση σε αρχεία
    - Μόνο εξουσιοδοτημένοι χρήστες μπορούν να προσπελάσουν συγκεκριμένα αρχεία.

Με μια ματιά…

Πως οργανώνουμε, όμως, ένα file system;;;

Οργάνωση ενός File System

- **Directory Service (Υπηρεσία Καταλόγου)**
  - **Directory Module**: Μετάφραση υψηλού επιπέδου ονόματα (filenames) σε File IDs
  - **Access Control Module**: έλεγχος άδειας πρόσβασης
- **File Service (Υπηρεσία Αρχείων)**
  - **File Module**: File IDs -> «files» (δομές που καθορίζουν πού βρίσκονται τα δεδομένα)
  - **File Access Module**: Προσπέλαση αρχείου
- **Block Service (Υπηρεσία Αποθήκευσης)**
  - **Block Module**: Διαχείριση χώρου στον δίσκο
  - **Device Module**: disk I/O & buffering



Block Service (bottom layer)

- Διαχειρίζεται **disk blocks** (μονάδες αποθήκευσης του δίσκου)
  - **Allocates blocks to files** (αναθέτει blocks σε αρχεία)
  - **Frees blocks** (ελευθερώνει blocks)
  - **Accesses** (προσπελαύνει) **blocks**
  - **Updates** (ενημερώνει) **blocks**

File Service (middle layer)

- Υλοποιεί έναν **επίπεδο χώρο ονομάτων** (**flat namespace**).
  - Πρέπει να μπορεί να προσδιορίζει την ταυτότητα του ζητούμενου αρχείου μονοσήμαντα
  - **Unique File IDs – UFIDs** (Μοναδικές ταυτότητες): Βάσει ενός UFID μπορεί να εντοπιστεί η μετα-πληροφορία του αντίστοιχου αρχείου.

- **File = Data + Attributes** (**δεδομένα + ιδιοχαρακτηριστικά**).
  - Attributes = Metadata (δηλ. πληροφορία για την πληροφορία).
  - Συνήθως περιγράφουν το μέγεθος, τον ιδιοκτήτη, τη λίστα ελέγχου πρόσβασης, κ.τ.λ.
  - Μερικά είναι προσπελάσιμα από τους χρήστες.

- Δίνει εντολές στο Block Service να **αναθέσει** / **ελευθερώσει** / **προσπελάσει** / **ενημερώσει** κάποια blocks.

Directory Service (top layer)

- Διαχειρίζεται καταλόγους (**directories**)
  - συνήθως σαν **ειδικά αρχεία**.
  - τα directories **αποθηκεύουν ζεύγη: <filename, UFID>**
- Ο directory server είναι client του file server.
- Διαχειρίζεται UFIDs, όταν δημιουργούνται αρχεία, που επιστρέφονται στους χρήστες.
- Έπειτα, οι χρήστες αναφέρονται στα αρχεία χρησιμοποιώντας τα UFIDs.

Directory Service (top layer)

- **Access Control** (**Έλεγχος πρόσβασης**)
  - Πριν επιστρέψει ένα UFID, πρέπει να ελέγξει αν ο χρήστης έχει δικαίωμα πρόσβασης ➔ **access control**
  - Τα UFIDs μπορεί να είναι δικαιώματα (access rights): δηλ. χρησιμοποιούνται από τους κατέχοντες σαν απόδειξη προς τον File Server του δικαιώματος προσπέλασης αρχείων, βάσει User IDs.

- **User Ids**: προσδιορίζουν την ταυτότητα χρηστών που προσπελαύνουν αρχεία.
  - Κάθε χρήστης σχετίζεται με ένα user id μετά από μια διαδικασία authentication (εξακρίβωση στοιχείων) (π.χ., passwords).
  - Μπορεί να υπάρχει ξεχωριστός authentication server.

Directory Service (top layer)

- **Access Control Lists** (ACLs)

Operation	Users
Read	Λίστα με χρήστες
Write	Λίστα με χρήστες

Επίσης μπορούν να δημιουργηθούν ομάδες χρηστών (με ομαδικά δικαιώματα πρόσβασης)

Σύνοψη:

- Δεδομένου ενός user id και ενός ονόματος αρχείου το directory service προσπελαύνει το ACL του αρχείου και, αν όλοι οι έλεγχοι εκτελεστούν ομαλά, **επιστρέφει ένα UFID** στον χρήστη.
- Σε όλες τις μετέπειτα αιτήσεις του ο χρήστης προσκομίζει μόνο το UFID για να αποδείξει το δικαίωμα πρόσβασής του στο αρχείο.

Directory Service

- (Ενδεικτικές) Λειτουργίες:
  - **AddName(DirUFID, Filename, UFID)**
    Προσθέτει το ζεύγος (Filename, UFID) στο directory
  - **UnName (DirUFID, Filename)**
    Το ζεύγος που περιέχει το “Filename" διαγράφεται από το DirUFID
  - **ReName (DirUFID, OldName, NewName)**
    Αλλάζει το ζεύγος <OldName,UFID> → <NewName,UFID>
  - **LookUp(DirUFID, Filename, Access Mode, Userld, &UFID)**
    - Σε επιτυχία ελέγχου πρόσβασης, επιστρέφει το UFID του filename
  - **GetName(DirUFID, pattern)**
    Επιστρέφει όλα τα ονόματα του DirUFID που ταιριάζουν στο "pattern".

Directory Service

- Το Directory service πρέπει επίσης να μπορεί να **αλλάζει τις ιδιότητες** ενός αρχείου -- π.χ., το ACL, για να υποστηρίζει **chmod**...
  - Επίσης πρέπει να παρέχει λειτουργίες για να εξετάζονται τα attributes.

- Το directory service είναι ο ιδιοκτήτης όλων των directories.
  - Για **shared** (**κοινόχρηστα**) **directories** πρέπει να δηλώσουμε έναν χρήστη σαν ιδιοκτήτη και μια στρατηγική access-permission.

Προσοχή!

- Η οργάνωση ενός file system σε ξεχωριστά επίπεδα μπορεί να δημιουργήσει προβλήματα, επειδή κάποιες λειτουργίες ΔΕΝ εκτελούνται ατομικά!!

- Παράδειγμα 1: **Δημιουργία νέου αρχείου**
  - καλούμε το **Create()** system call (λειτουργία του File Service)
  - κατόπιν το **AddName()** (λειτουργία του Directory Service)
  - Έτσι, σφάλματα/αποτυχίες δημιουργούν προβλήματα (π.χ., χαμένα αρχεία)
  - Αν, λόγω βλάβης, δεν εκτελέσει το δεύτερο βήμα τότε έχουμε ένα μη-προσπελάσιμο αρχείο το οποίο δεν μπορούμε να διαγράψουμε.

- Παράδειγμα 2: **Διαγραφή αρχείου**
  - καλούμε το **Delete()** system call (λειτουργία του File Service)
  - κατόπιν το **UnName()** (λειτουργία του Directory Service)

File Service

- (Ενδεικτικές) λειτουργίες για τα περιεχόμενα ενός αρχείου:
  - **Create(&UFID)**
    - Δημιουργεί ένα νέο (άδειο) αρχείο και επιστρέφει ένα UFID
  - **Delete(UFID)**
    - Διαγράφει το αρχείο
  - **Length(UFID)**
    - Επιστρέφει το μήκος του αρχείου
  - **Read(UFID, number, &buffer)**
    - Διαβάζει "number" bytes αρχίζοντας από τη θέση που δείχνει ο **δείκτης** του αρχείου UFID και τα τοποθετεί στο "buffer".
  - **Seek(UFID, position)**
    - Τοποθετεί τον **δείκτη** του αρχείου στη θέση “position”.
  - **Write(UFID, buffer)**
    - Γράφει τα δεδομένα του "buffer" στο αρχείο UFID στη θέση που δείχνει ο **δείκτης** του αρχείου.

File Service

(Ενδεικτικές) Λειτουργίες για τα attributes:

- **SetAttributes(UFID, attributes)**
- **GetAttributes(UFID, &attributes)**

Σε κατανεμημένα file systems, τα **UFIDs** πρέπει να είναι:

- **μοναδικά** (μεταξύ **όλων των files** και **όλων των κόμβων** που υλοποιούν το F.S.)
- **δύσκολο να "μαντευτούν"**

48 bits	32 bits	32 bits
server host id	file number	random number

server host id ➔ Internet address, (μπορεί να είναι μικρότερο για ένα ethernet)

random number ➔ για να είναι δύσκολο να μαντευτεί.

File Service

- Εντοπίζοντας δεδομένα αρχείων
  - **Input**: **UFID**
  - **Output**: **Τοποθεσία του file index**
  - Για τον σκοπό αυτό έχουμε το **File Location Map**!

- Στην πραγματικότητα κάθε αίτηση που έρχεται στο file service περιέχει ένα **UFID** και ένα **offset**, π.χ., **read(UFID, 1500, &buffer)**
  a) UFID ➔ εύρεση του block του σχετικού file index
  b) offset ➔ εύρεση του ζητούμενου block του αρχείου
  - (a)+(b): μετάφραση σε 2 βήματα

- Το **File Location Map** κάνει το (a)
  - Εξ αιτίας της ύπαρξής του τα αρχεία μπορούν να μεταφερθούν εύκολα.

- Το **File Index** κάνει το (b)

File Service

- **Υλοποίηση**
  - Απαιτείται **δυναμική ανάθεση** (**allocation**) των **blocks** του δίσκου ώστε να επιτρέπονται δυναμικές αλλαγές στα αρχεία.
  - Ένα αρχείο γενικά καταλαμβάνει μη συνεχόμενα disk blocks (Γιατί;)

- **File Index**
  - Το file service διατηρεί μια δομή δεδομένων, το **File Index**. Το file index συνήθως περιέχει τα attributes του αρχείου και μια ακολουθία από δείκτες στα blocks του δίσκου που περιέχουν τα δεδομένα του αρχείου.
  - Το file index πρέπει να υποστηρίζει **σειριακή και τυχαία πρόσβαση** (**sequential + random access**) ενός file.

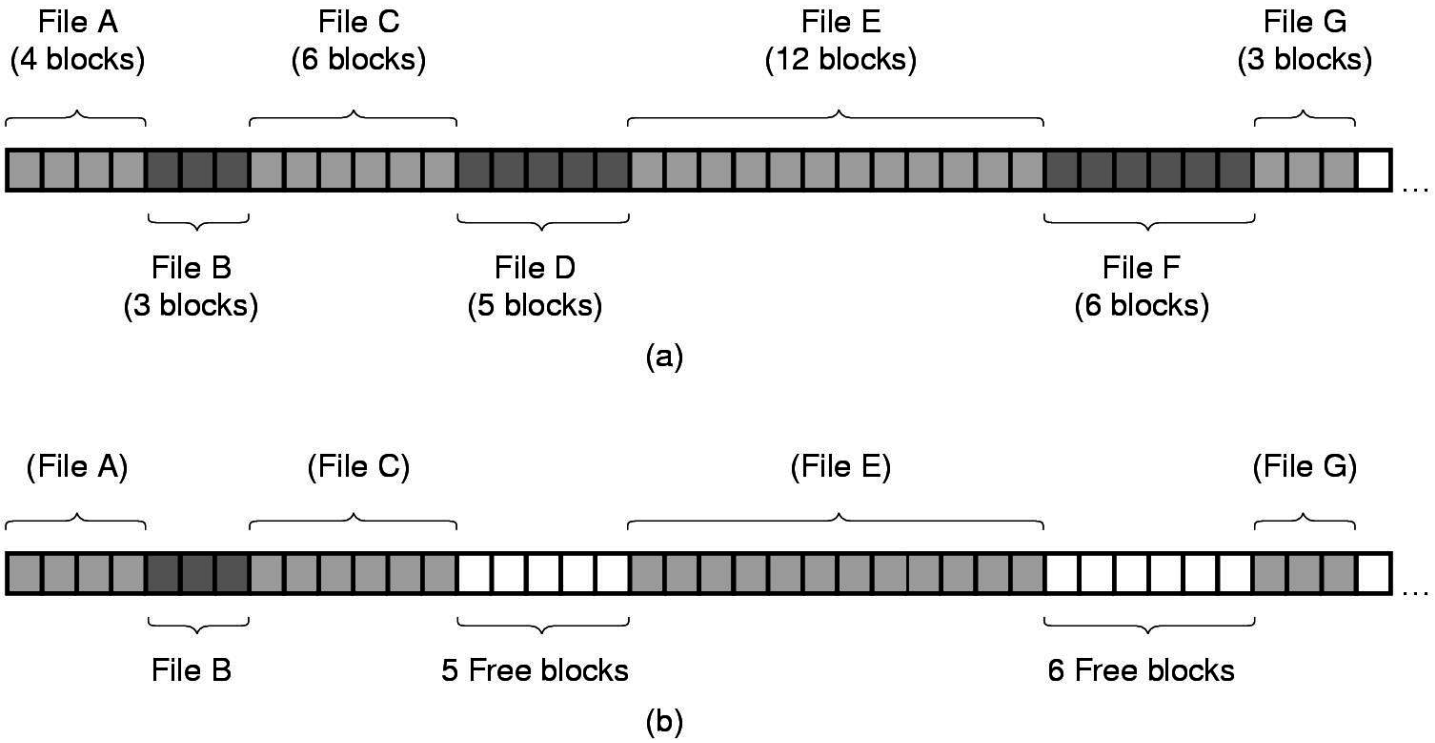
Τρόποι ανάθεσης blocks

- **Block Allocation** (**Ανάθεση blocks**)
  - Με ποιον τρόπο αναθέτουμε blocks σε ένα αρχείο;

- Υπάρχουν 3 βασικές στρατηγικές ανάθεσης:
  - Συνεχόμενη ανάθεση (contiguous allocation)
  - Ανάθεση βασιζόμενη σε συνδεδεμένες λίστες (linked-list allocation)
  - Δεικτοδοτημένη ανάθεση (indexed allocation)

Contiguous Allocation

- Όταν ένα αρχείο χρειάζεται N disk blocks, τότε N συνεχόμενα blocks ανατίθενται στο αρχείο αυτό.



Contiguous Allocation

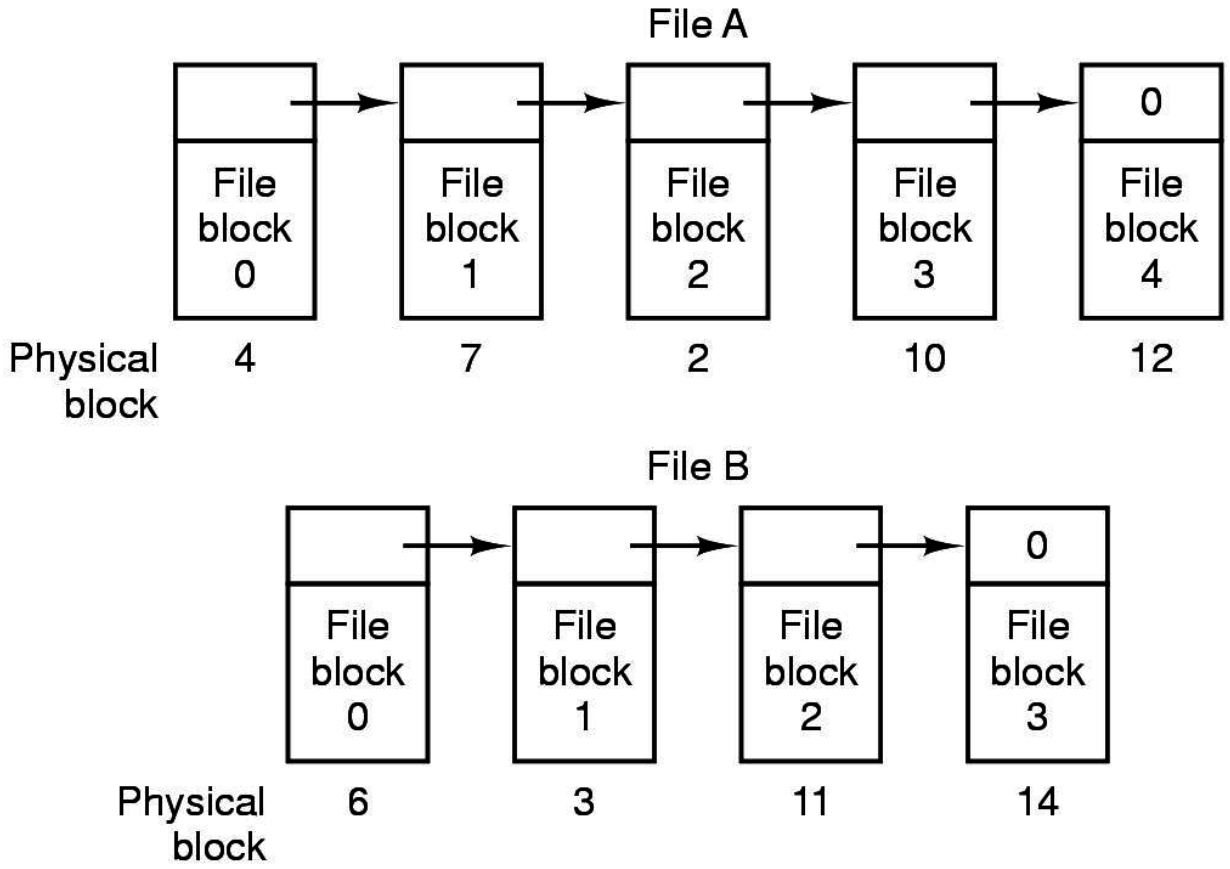
- **Συν:**
  - **Απλότητα**: Ο πιο **απλός** τρόπος ανάθεσης.
  - **Γρήγορη πρόσβαση** στο αρχείο. Για να διαβαστεί ολόκληρο το αρχείο πληρώνουμε το κόστος για το **disk seek delay** (καθυστέρηση μετακίνησης κεφαλής) και για το **rotation delay** (καθυστέρηση περιστροφής δίσκου) μόνο μια φορά.
  - Αυτό είναι σημαντικό εξ αιτίας του σχετικά μεγάλου κόστους!

- **Πλην:**
  - Πως γνωρίζουμε εκ των προτέρων το μέγιστο μέγεθος ενός αρχείου;
  - Fragmentation: όταν τα αρχεία διαγράφονται παραμένει ελεύθερος χώρος κατακερματισμένος σε μικρά τμήματα, που δεν είναι αρκετά μεγάλα να «χωρέσουν» άλλα αρχεία.

- Λύση: **compaction** (**σύμπτυξη**)
  - Αυτό στοιχίζει πάρα πολύ (disk_to_disk copying). Μπορεί όμως να γίνει "off-line" δηλ. όταν το σύστημα δεν χρησιμοποιείται. (π.χ., τη νύχτα).

Linked-List Allocation

- Τα αρχεία είναι μια λίστα από blocks, με μερικά bytes του κάθε block να δείχνουν στο επόμενο block της λίστας.



Linked-List Allocation

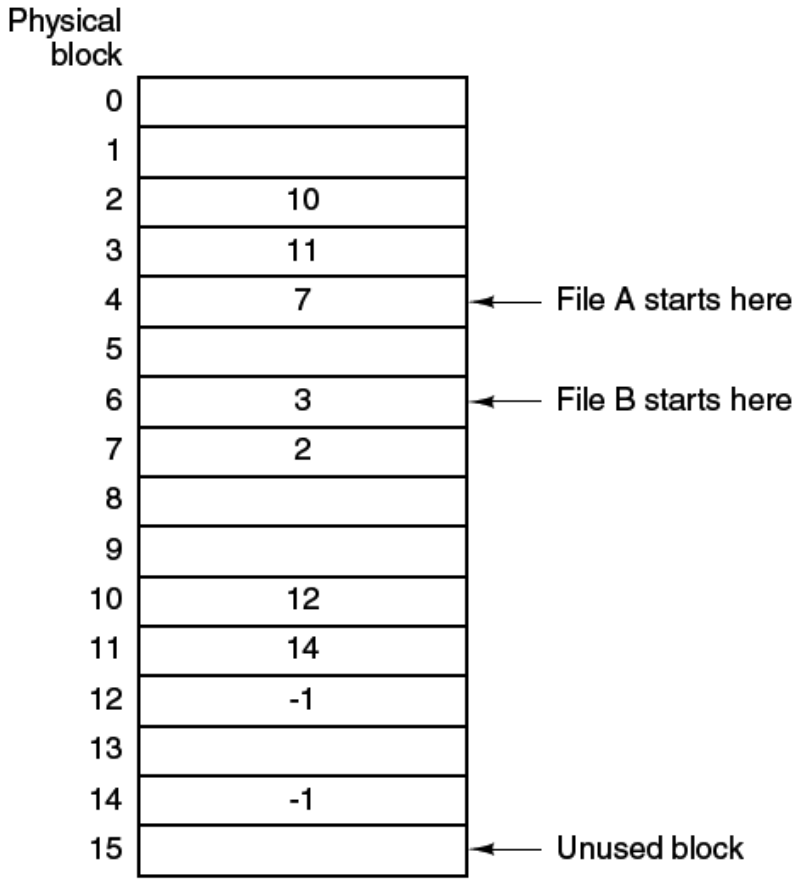
- **Συν:**
  - Αποφεύγεται το πρόβλημα κατακερματισμού.
  - Επίσης η κάθε εγγραφή καταλόγου αποθηκεύει μόνο τη διεύθυνση του 1ου block του αρχείου.

- **Πλην:**
  - Το βασικό πρόβλημα αυτής της μεθόδου είναι ότι όταν επιθυμείται **τυχαία προσπέλαση** (**random access**), η απόδοση του συστήματος είναι πολύ κακή!
  - Π.χ., για να προσπελαστεί το N-οστό block πρέπει πρώτα να γίνουν N-1 επιπλέον disk I/Os.

- **Λύση για γρήγορο random access;;;**
  - Χρήση ενός index στην RAM, που περιέχει όλους τους δείκτες στα blocks.

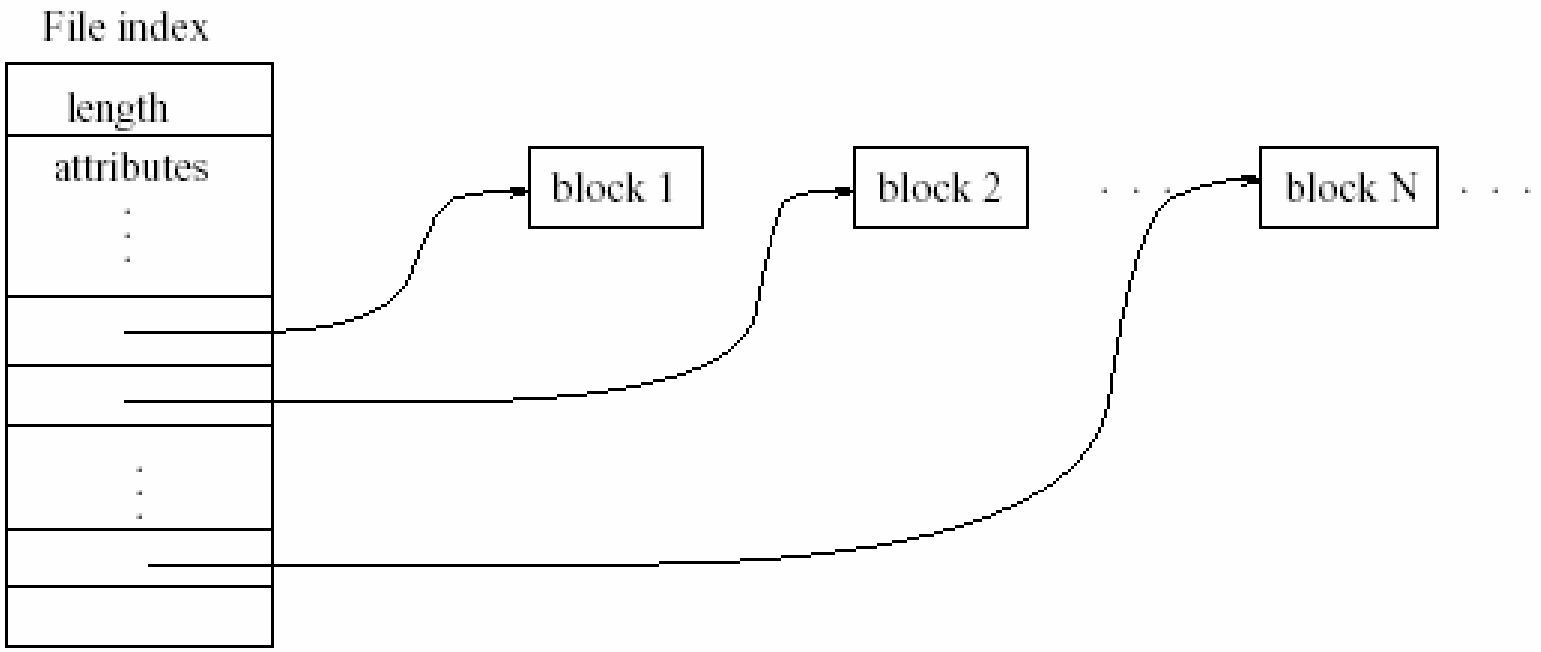
Indexed Allocation

- Χρήση ενός index στην RAM, που περιέχει όλους τους δείκτες προς τα blocks.
- Είδη Indexed Allocation
  - **Per file index:** ένα index ανά αρχείο (Unix)
  - **Global index:** ένα index για όλα τα αρχεία (MS-DOS)
- **+/-**
  - Με global index, το index βρίσκεται πάντα στη RAM
  - Επιπλέον disk I/Os ελαχιστοποιούνται
  - Όμως το global index καταναλώνει πολλή RAM γιατί διαθέτει μια εγγραφή για κάθε block του δίσκου.



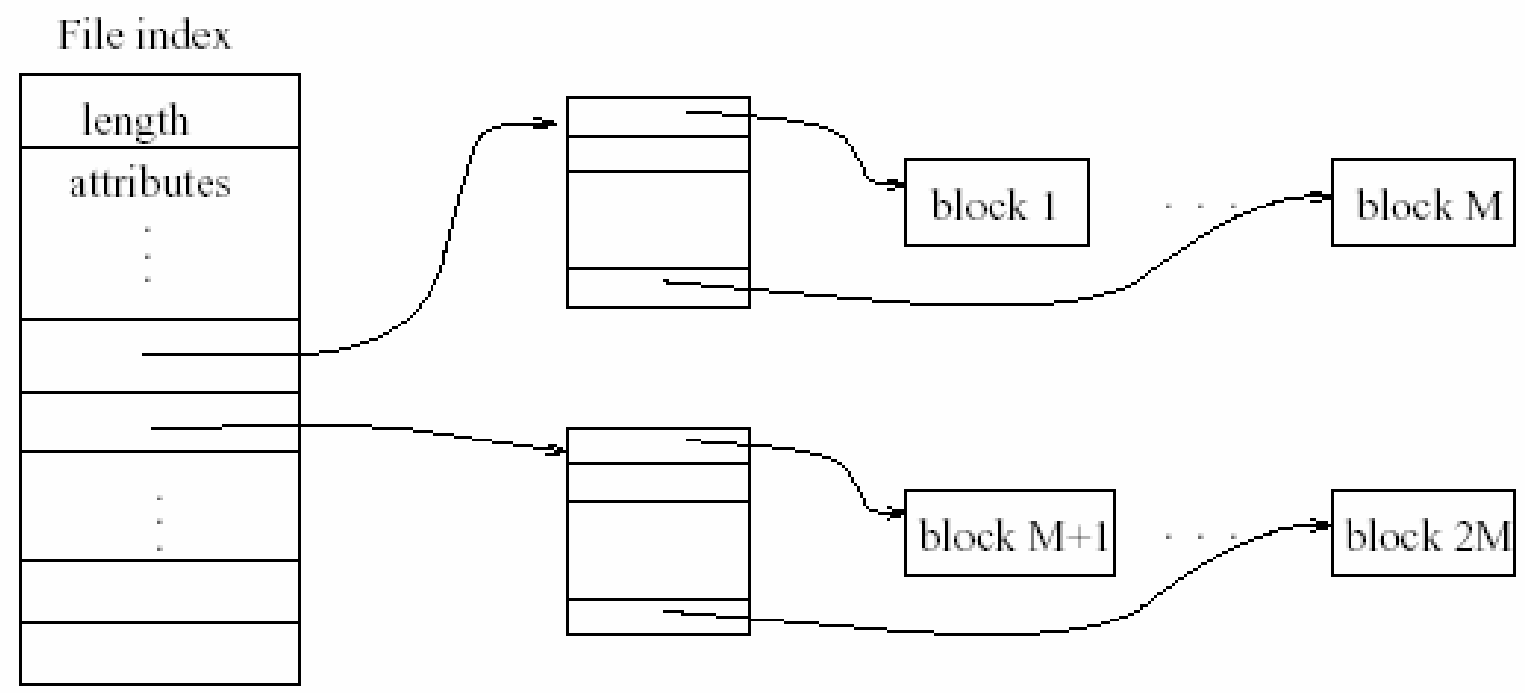
Indexed Allocation

Παράδειγμα Indexed Allocation



Indexed Allocation

- Μπορούμε να έχουμε file index πολλαπλών επιπέδων
  - π.χ. file index 2 επιπέδων:



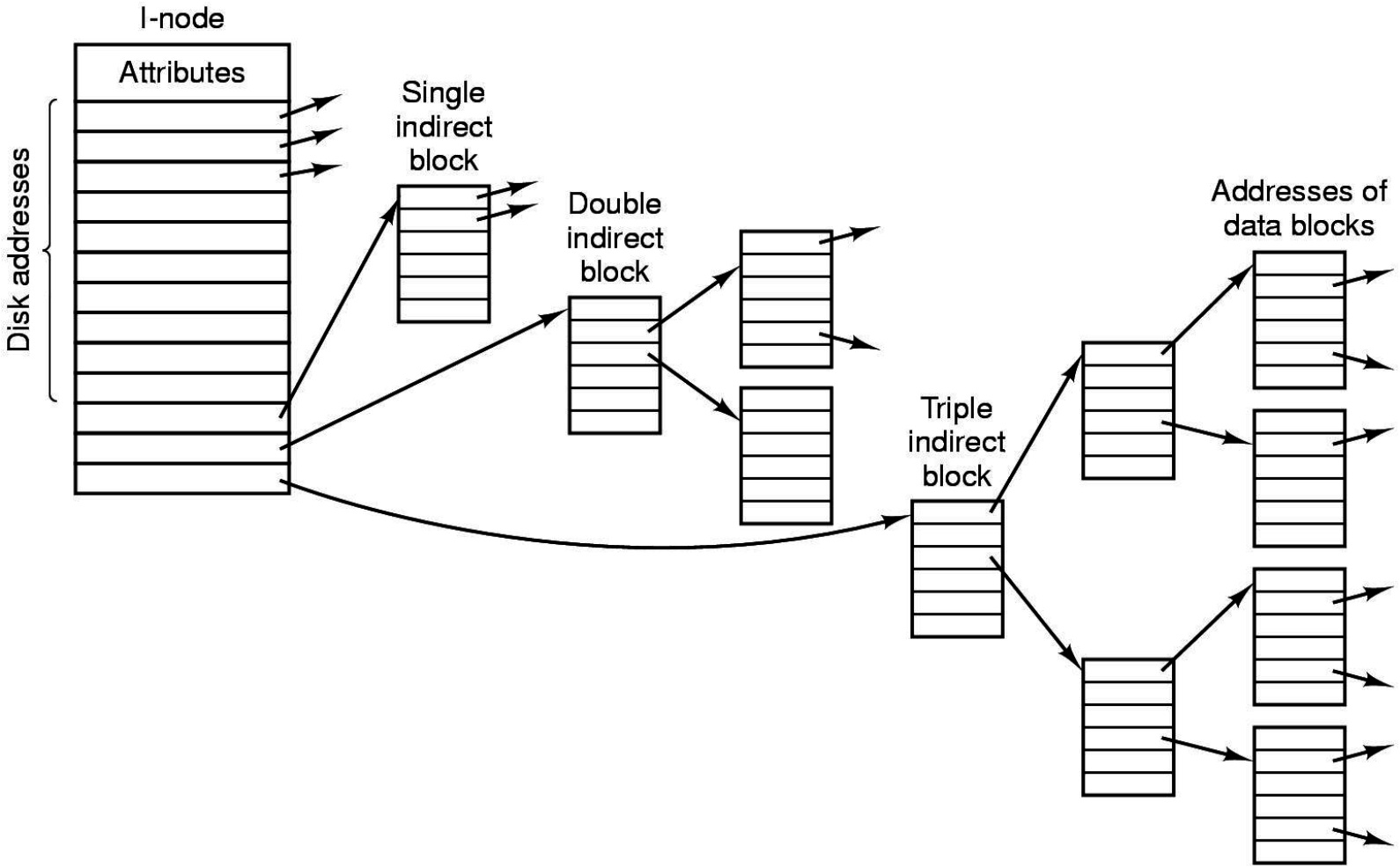
Indexed Allocation

- Ένα παράδειγμα per-file index είναι τα inodes που χρησιμοποιεί το UNIX.

- Το inode περιέχει:
  - τα attributes
  - 10 δείκτες για τα πρώτα 10 blocks του αρχείου
  - 1 δείκτη που δείχνει σ' ένα disk block (single indirect block) που με τη σειρά του δείχνει στα επόμενα 256 disk blocks του file.
  - 1 δείκτη σε double indirect block
  - 1 δείκτη σε triple indirect block

File Service: **i-node**

Το UNIX έχει μια δομή που ονομάζεται **i-node** (file index).



Block Service

- Γενικά:
  - **δέσμευση** και **αποδέσμευση** blocks του δίσκου
  - **μεταφορά δεδομένων** από και προς τα blocks του δίσκου

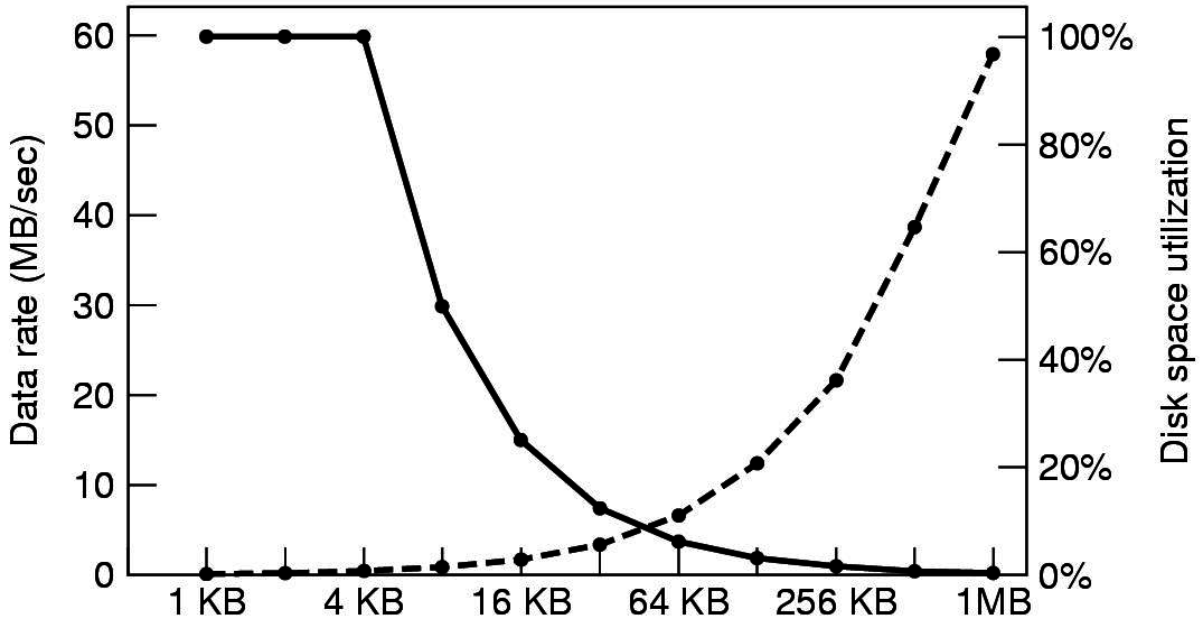
- Block pointer = disk_id + διεύθυνση στον δίσκο

Block Service

- (Ενδεικτικές) Λειτουργίες
  - **AllocateBlock (&blockptr)**
    - δεσμεύει ένα νέο block στον δίσκο και επιστρέφει έναν δείκτη σε αυτό το block.
  - **FreeBlock (blockptr)**
    - ελευθερώνει το block που δείχνεται από τον blockptr.
  - **PutBlock (blockptr, &buffer)**
    - γράφει τα περιεχόμενα του buffer στο disk block που δείχνεται από τον blockptr
  - **GetBlock (blockptr, &buffer)**
    - διαβάζει τα περιεχόμενα του block που δείχνεται από τον δείκτη blockptr και τα αποθηκεύει στον buffer.
- Οι παραπάνω λειτουργίες υλοποιούνται χρησιμοποιώντας περισσότερο στοιχειώδεις λειτουργίες του δίσκου (primitive disk ops), …παρέχονται από το device module.

Block Service

- **Tradeoff του block size** (Μονάδας Αποθήκευσης Πληροφορίας)
  - **Χώρος**: μειωμένη χρήση αποθηκευτικού χώρου στον δίσκο για μεγάλα blocks έναντι καλύτερης χρήσης αποθηκευτικού χώρου στον δίσκο με μικρότερα blocks → **Πάρα πολλά αρχεία είναι πολύ μικρά!**
  - **Ταχύτητα**: Η μεταφορά των δεδομένων μεταξύ δίσκου -- κύριας μνήμης είναι >1000 φορές πιο γρήγορη από το κόστος αναζήτησης + κόστος περιστροφής
    - όσο λιγότερα blocks αναζητούμε τόσο το καλύτερο.



Block Service

- Κάποια συστήματα χρησιμοποιούν πολλαπλά μεγέθη block (π.χ., 4.2. BSD):
  - Ενα block χωρίζεται σε τμήματα (fragments).
  - Ένα αρχείο των N bytes αποτελείται από (N / b) blocks και από κανένα ή ένα τμήμα από κάθε μέγεθος (b/2, b/4, b/8) (b είναι ο αριθμός των bytes ανά block)
  - Εάν b είναι 8K, πολλά μικρά αρχεία μπορούν να αποθηκευτούν σ' ένα απλό block (χρησιμοποιώντας τα τμήματά του).

- Αυτός ήταν ο κύριος λόγος της επιτάχυνσης του συστήματος αρχείων του BSD από 2-5% σε 30-47% της ονομαστικής «ταχύτητας» του δίσκου (raw disk bandwidth).

Επικοινωνία File Service-Block Service

- **Create:**
  - παράγει UFID +
  - δεσμεύει χώρο για το file index +
  - αρχικοποιεί attributes
  - προσθέτει μια νέα είσοδο (entry) στο File Location Map και
  - επιστρέφει το UFID.
- **Delete:**
  - σβήνει την είσοδο από το File Location Map
  - ελευθερώνει τα blocks του αρχείου
  - ελευθερώνει τα blocks που δεσμεύει το file index
- **Read:**
  - χρησιμοποιείται η GetBlock για κάθε block που χρειάζεται
- **Write:**
  - AllocateBlock (στο τέλος του αρχείου) + PutBlock ή
  - GetBlock (overwrite) + PutBlock

Memory-mapped files

- Πολλοί υποστηρίζουν ότι το παραπάνω interface του filesystem είναι... "άβολο". Μια εναλλακτική λύση είναι τα memory-mapped files.

- Ο χρήστης μπορεί να καλέσει το Map (filename, virtual-address) όπου το virtual-address είναι μια μεταβλητή προγράμματος (π.χ., char *filebuffer).

- Το Λ.Σ., εκτελώντας αυτή την εντολή, χρησιμοποιεί τους πίνακες που κρατούν τις διευθύνσεις των disk blocks που αποτελούν το αρχείο και ενημερώνει τα disk blocks που αποτελούν τον χώρο εναλλαγής (“swap space”) για το process και συγκεκριμένα τον χώρο που αφορά στη μεταβλητή filebuffer.

Memory-mapped files

- το swap space για το virtual address space στο οποίο χαρτογραφείται το αρχείο περιέχει τα disk blocks του αρχείου.
- τα page faults εξυπηρετούνται προσπελαύνοντας τα κατάλληλα disk blocks (από το swap space)
- το process προσπελαύνει ένα αρχείο όπως οποιαδήποτε μεταβλητή/δομή δεδομένων του προγράμματός του (π.χ. με ένα for loop)
  - for (i>=1) do {… file[i]:= …. }

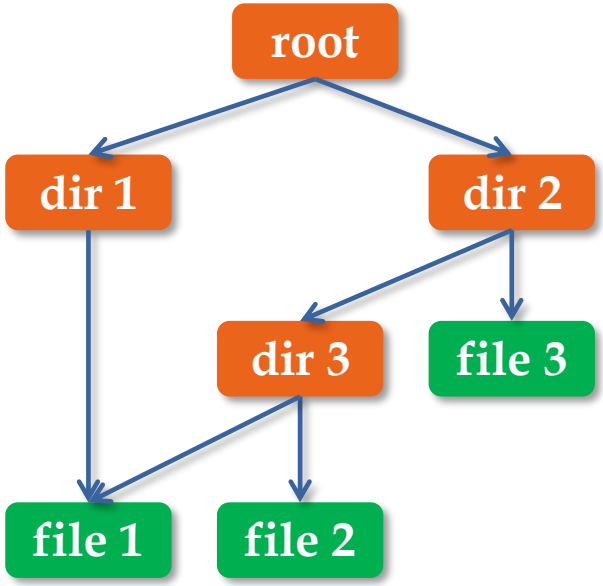
Memory-mapped files

- Η χρήση memory-mapped files, δημιουργεί δυνητικά προβλήματα **διαμοιρασμού** (**sharing**) και **συνέπειας** (**consistency**).

- Παράδειγμα
  - Ένα process P1 έχει ενημερώσει ένα αρχείο, αλλά τα σχετικά blocks του αρχείου δεν έχουν αποθηκευτεί στον δίσκο ακόμα (δηλ. δεν έγιναν page out).
  - Ένα άλλο process P2 τότε προσπελαύνει το ίδιο αρχείο χρησιμοποιώντας για παράδειγμα το κανονικό File System interface (open()/read(),…)
  - Δεν θα δει την πιο πρόσφατη πληροφορία!

Directories

- Τα directories συνήθως είναι ιεραρχικά
  - Υπάρχει ένα root directory
  - Κάθε directory μπορεί να περιέχει αρχεία ή/και άλλα directories
- Υλοποιούνται σαν αρχεία (**directory files**) που περιέχουν **directory entries**
- Κάθε directory entry είναι ένα ζεύγος **<filename,UFID>**
  - το UFID προσδιορίζει (δομές που κρατούν) τις διευθύνσεις των disk blocks που αποτελούν το αρχείο καθώς και τα attributes του
- Σε file systems του Unix, το UFID είναι ουσιαστικά το inode του σχετικού αρχείου



Directories

- Σε ιεραρχικά συστήματα υπάρχουν 2 τρόποι ονομασίας ενός αρχείου
  - **Absolute** path name (**απόλυτη**)
  - **Relative** path name (**σχετική**)
- Το απόλυτο όνομα είναι μοναδικό για κάθε αρχείο και πάντα αρχίζει με το root directory
- Το σχετικό όνομα σχετίζεται με την έννοια του **current** (ή **working**) **directory**.
  - Αν το current directory είναι το **/usr/bin**, τότε τα ονόματα **make** και **/usr/bin/make** αναφέρονται στο ίδιο αρχείο
- Κάθε process φυσικά έχει το δικό της current directory
  - δηλ. είναι μια per-process μεταβλητή
  - Αν για το process 1 το current dir. είναι **/usr/bin** και για το process 2 current dir είναι **/etc** τότε το σχετικό όνομα **foo** αναφέρεται σε διαφορετικά αρχεία, ανάλογα με το process.

Directories

- Οι πιο συνήθεις λειτουργίες που παρέχονται από directory services είναι: opendir(), closedir(), create(), delete(), readdir(), rename(), link(), unlink().

- **readdir()**
  - Το readdir() είναι πλεονάζον, μιας και θα μπορούσε να αντικατασταθεί από κλήσεις στο read() για το σχετικό directory file.
  - Όμως αυτό θα απαιτούσε από το κάθε user program να γνωρίζει την εσωτερική δομή του directory file κάθε file system!!
  - Το readdir() απαλλάσσει τον προγραμματιστή απ' αυτή την ανάγκη, προσφέροντάς του ένα γενικό API για όλα τα ιεραρχικά file systems
  - Το readdir(), λοιπόν, επιστρέφει την επόμενη εγγραφή του directory, ή NULL αν φτάσαμε στο τέλος

- **link()**
  - Συνδέει ένα inode με ένα όνομα αρχείου. Μάλιστα επιτρέπεται σε πολλά directories να "περιέχουν" το ίδιο αρχείο - έτσι π.χ., επιτρέπεται ο διαμοιρασμός του αρχείου από διαφορετικούς χρήστες που το τοποθετούν στο δικό τους directory.

- **unlink()**
  - Διαγράφει ένα directory entry. Αν αυτό ήταν το μοναδικό dir. entry που αναφέρεται στο αρχείο τότε διαγράφεται επίσης το ίδιο το αρχείο.

Pathname translation

- Στο Unix τα directory entries είναι ζεύγη **<filename, inode#>**
  - Δηλαδή το UFID είναι ουσιαστικά ο αριθμός του inode.

- **Pathname translation**
  - Η διαδικασία που δέχεται σαν είσοδο ένα pathname (π.χ., **/d1/d2/d3/foo**) και παράγει σαν έξοδο το inode# του αρχείου, λέγεται **pathname translation** (**μετάφραση ονόματος**).

Pathname translation

1. Το inode για το root directory "/" ανακτάται από τον δίσκο. (Αυτό το inode βρίσκεται σε μια γνωστή και μη τροποποιήσιμη διεύθυνση στον δίσκο).
   - Απ' το inode του root βρίσκονται τα disk blocks που περιέχουν το αρχείο (directory file) του root directory. Δηλαδή το αρχείο που περιέχει τα directory entries.
   - Αυτά τα blocks ανακτώνται και ψάχνονται για την εγγραφή "d1". Όταν βρίσκεται η εγγραφή για το "d1" τότε βρίσκεται και το inode# του d1.
2. Το inode για το "d1" ανακτάται απ' το δίσκο.
   - Τα disk blocks του dir file "d1" ανακτώνται και ψάχνονται για την εγγραφή "d2"….
3. Τέλος το inode & disk blocks του "d3" ανακτώνται και ψάχνονται για την εγγραφή "foo".
   - Έτσι βρίσκεται το inode# για το αρχείο «foo» το οποίο ανακτάται και τοποθετείται στη μνήμη (σ' ένα cache ειδικό για inodes που λέγεται inode table).
4. Αφού τοποθετηθεί το inode στο inode table του πυρήνα ο file descriptor πίνακας του process ενημερώνεται.
5. Μια εγγραφή του χρησιμοποιείται για έναν δείκτη ακολουθώντας τον οποίο μπορεί ο kernel να βρει το inode στο inode table.
6. Τέλος επιστρέφεται ο δείκτης της εγγραφής του file descriptor πίνακα...

Links

- Για να διευκολυνθεί ο διαμοιρασμός αρχείων, συχνά ένα αρχείο εμφανίζεται σε πολλούς διαφορετικούς καταλόγους (π.χ. σ' έναν κατάλογο για κάθε χρήστη που το διαμοιράζεται).
- Αυτό επιτυγχάνεται με **links** που συνδέουν ένα πεδίο καταλόγου μ' ένα αρχείο σε κάποιον άλλον κατάλογο.
- Αν τα πεδία καταλόγων περιείχαν τις διευθύνσεις των blocks στον δίσκο, τότε στον κατάλογο που ορίζεται το link σ' ένα αρχείο F1 θα πρέπει να γίνει ένα αντίγραφο των διευθύνσεων των blocks του F1. Αν αυτά αλλάξουν (π.χ. append()) τότε προκύπτει πρόβλημα διαχείρισης αντιγράφων.
- Στο Unix δεν έχουμε τέτοιο πρόβλημα. Γιατί;

Links

- **Hard links** - Στο UNIX το πρόβλημα λύνεται ως εξής:
  - Τα πεδία καταλόγων δεν περιέχουν διευθύνσεις των blocks του δίσκου αλλά τον **αριθμό του inode** του σχετικού αρχείου, το οποίο περιέχει τις **διευθύνσεις των blocks**.
  - Έτσι πολλά directories μπορούν να αναφέρονται στο ίδιο inode, οπότε το αρχείο αυτό «περιέχεται» σε όλα αυτά τα directories (που όλα «δείχνουν» στο **ίδιο αντίγραφο**).
  - Όλα τα links αυτού του αρχείου είναι **ισοδύναμα**. Δηλαδή δεν είναι κανένα το «επίσημο» αντίγραφο, και τα υπόλοιπα «δευτερεύοντα».
  - Γίνεται με την εντολή **ln**. Π.χ., **ln file1 file2**

- **Symbolic/Soft links** - Μια άλλη λύση βασίζεται στην έννοια των «συμβολικών συνδέσμων»
  - Όταν δημιουργείται ένα link L1 στο αρχείο F1, το L1 είναι ένα ειδικό αρχείο τύπου **link**.
  - Τα περιεχόμενα αρχείων τύπου link είναι μόνο το **pathname του αρχείου στο οποίο αναφέρονται**.
  - Όπως είναι προφανές, τα symbolic links προσθέτουν παραπάνω κόστος στο pathname translation
  - Γίνονται με την εντολή **ln -s**. Π.χ., **ln -s file1 file2**

Links

- Τι περιορισμοί υπάρχουν στη χρήση hard και symbolic links;

- **Filesystem boundaries**
  - Τα **hard links** μπορούν να χρησιμοποιηθούν μόνο **εντός του ίδιου filesystem**.
  - Για cross-over σε άλλο filesystem χρησιμοποιούμε υποχρεωτικά **symbolic links**.

- **Directories**
  - Τα **hard links** μπορούν να χρησιμοποιηθούν **μόνο σε αρχεία**.
  - Δηλαδή **δεν μπορούμε** να κάνουμε **hard link** σε κάποιο **directory**.
  - Συνεπώς για link σε κάποιο directory, χρησιμοποιούμε **symbolic link**.

Filesystem Cache

- Όπως έχουμε πει, κάθε πρόσβαση στην Κ.Μ. κοστίζει **< 1μs**, ενώ κάθε πρόσβαση στον δίσκο κοστίζει **~ 8ms**.

- Το FS θα έχει καλύτερη απόδοση όσο πιο λίγες φορές χρειάζεται disk I/O!!!

- **Cache memory** (**Προσωρινή/Κρυφή Μνήμη**)
  - Η επίτευξη του στόχου αυτού βασίζεται στην έννοια του filesystem buffer ή cache.
  - Η κρυφή μνήμη είναι ένα τμήμα της Κ.Μ. (στον πυρήνα) το οποίο αποθηκεύει προσωρινά μερικά blocks του δίσκου.

Προσωρινή (Κρυφή) Μνήμη

- Όταν κάποια διεργασία ζητεί πρόσβαση σε κάποιο block, ο πυρήνας
  - **φορτώνει** πρώτα αυτό το block στην **κρυφή μνήμη** και
  - μετά το δίνει στη διεργασία.
- Οι μεταγενέστερες αιτήσεις (από την ίδια ή άλλες διεργασίες) για το ίδιο block **θα εξυπηρετηθούν από την κρυφή μνήμη**.
- Το disk I/O αποφεύγεται!
- Έτσι, κάθε φορά που ζητείται κάποιο block, ο kernel πρώτα ψάχνει την κρυφή μνήμη και μόνο αν δεν το βρει απευθύνεται στον δίσκο.

Προσωρινή (Κρυφή) Μνήμη

- Φυσικά, απαιτείται κάποιος αλγόριθμος αντικατάστασης έτσι ώστε να λειτουργεί η πιο πάνω διαδικασία ακόμα και όταν γεμίζει η κρυφή μνήμη.
- Αυτός ο αλγόριθμος είναι συνήθως LRU (με μερικές αλλαγές).
- Πριν ένα block της Buffer Cache "φιλοξενήσει" ένα καινούργιο block του δίσκου, θα πρέπει το αρχικό block να εγγραφεί στον δίσκο αν έχει ενημερωθεί όσο ήταν στην κρυφή μνήμη (δηλ. αν είναι dirty).
- **Τι σας θυμίζει αυτό;;;**
  - **Θυμηθείτε!**
  - Αυτή είναι η ανάλογη ενέργεια που κάνει το **paging** σύστημα, που γράφει μια σελίδα στο swap area πριν το ίδιο page frame φιλοξενήσει κάποια καινούργια σελίδα.

Προσωρινή (Κρυφή) Μνήμη

- Οι αλλαγές στον αλγόριθμο LRU χρειάζονται:
  - για να αποφευχθούν όσο το δυνατόν περισσότερα disk I/Os... (ζήτημα κόστους)
  - για να αποφευχθούν προβλήματα inconsistency (ασυνέπειας) του filesystem ή χάσιμο δεδομένων όταν συμβούν βλάβες.

- **Ζήτημα ασυνέπειας** (**inconsistency**):
  - Όταν ένα block ενημερώνεται, αυτό πρώτα γίνεται στην κρυφή μνήμη.
  - Αν συμβεί κάποια αστοχία/σφάλμα τότε η ενημέρωση ‘χάνεται’.

- Απαιτείται μια πολιτική εγγραφής (write policy), που να γράφει τα blocks στον δίσκο έγκαιρα.

Προσωρινή (Κρυφή) Μνήμη

- Όταν κάποιο block στην κρυφή μνήμη περιέχει **ευαίσθητα δεδομένα**, όσον αφορά στη **συνέπεια του συστήματος**, τότε όταν αυτό το block ενημερωθεί εγγράφεται **αμέσως** στον δίσκο (σύγχρονα ή ασύγχρονα).

- Επίσης συνήθως υπάρχει κάποιο άνω όριο, όσον αφορά τον χρόνο κατά τον οποίο ένα ενημερωμένο block παραμένει στην κρυφή μνήμη χωρίς να εγγραφεί στον δίσκο.

- Για να μην χαθούν δεδομένα όταν συμβούν σφάλματα και βλάβες, πολλά συστήματα παρέχουν ειδικά system calls (π.χ., flush() ή SYNC) τα οποία γράφουν όλα τα ενημερωμένα blocks από την κρυφή μνήμη στον δίσκο.

- Σε πολλά «UNIX filesystems» (π.χ., NFS) υπάρχει ένα «πρόγραμμα παρασκηνίου» (background program) το οποίο εκτελείται κάθε *T* (π.χ., 30) sec και καλεί τη SYNC.
  - Έτσι στη χειρότερη περίπτωση μπορεί να χαθούν μόνο τα δεδομένα που γράφτηκαν τα τελευταία *T* secs.

Προσωρινή (Κρυφή) Μνήμη

- **Ευαίσθητα δεδομένα** θεωρούνται τα:
  - inode blocks,
  - directory blocks,
  - και γενικά blocks που δεν έχουν απλά δεδομένα χρηστών.

- Σκεφθείτε τι θα συμβεί αν το inode block ενός αρχείου ενημερωθεί και πριν εγγραφεί στο δίσκο, καταρρεύσει το σύστημα…
  - Tragic! ☺

- Ζήτημα κόστους
  - Όταν κάποιο block στην κρυφή μνήμη είναι μόνο μερικώς ενημερωμένο τότε τοποθετείται στο τέλος της LRU λίστας (καθυστερημένη εγγραφή)
  - Αφού δεν αναπληρώνεται αμέσως υπάρχει πιθανότητα όταν, μετά από χρόνο, αναπληρωθεί, να έχει ενημερωθεί ολόκληρο
  - Έτσι θα απαιτηθεί μόνο ένα disk I/O για όλες τις ενημερώσεις ενός block.

Προσωρινή (Κρυφή) Μνήμη

- Απαιτείται
  - Μηχανισμός γρήγορου εντοπισμού block στην Προσωρινή (Κρυφή) Μνήμη
  - Μηχανισμός διαχείρισης ελεύθερων block στην Προσωρινή (Κρυφή) Μνήμη
  - Σε συνεργασία με μηχανισμούς διαχείρισης σύγχρονων, ασύγχρονων και καθυστερημένων εγγραφών.

Filesystem Cache στο UNIX