WEEK 8

Disk I/O, examples: Linux, Windows File systems

Ch 11.5-11 [Stal 05] Ch 20.8 [DDC 04]

Ch 12.1-7 [Stal 05]

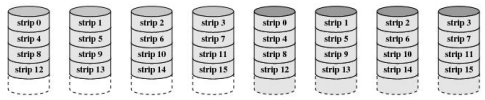
1

RAID -Redundant Array of Independent Disks (vai Inexpensive Disks)

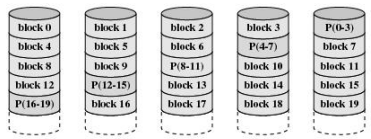
Ch 11.6 [Stal 05]

2

Mostly used: RAID 1 or RAID 5



(b) RAID 1 (mirrored)



(f) RAID 5 (block-level distributed parity)

3

RAID table

Tbl 11.4 [Stal 05]

Category	Level	Description	Disks required	Data availability	Large I/O data transfer capacity	Small I/O request rate
Striping	0	Nonredundant	N	Lower than single disk	Very high	Very high for both read and write
Mirroring	1	Mirrored	$2N$, $3N$, etc.	Higher than RAID 2, 3, 4, or 5; lower than RAID 6	Higher than single disk for read; similar to single disk for write	Up to twice that of a single disk for read; similar to single disk for write
Parallel access	2	Redundant via Hamming code	$N+m$	Much higher than single disk; higher than RAID 3, 4, or 5	Highest of all listed alternatives	Approximately twice that of a single disk
	3	Bit-interleaved parity	$N+1$	Much higher than single disk; comparable to RAID 2, 4, or 5	Highest of all listed alternatives	Approximately twice that of a single disk
Independent access	4	Block-interleaved parity	$N+1$	Much higher than single disk; comparable to RAID 2, 3, or 5	Similar to RAID 0 for read; significantly lower than single disk for write	Similar to RAID 0 for read; significantly lower than single disk for write
	5	Block-interleaved distributed parity	$N+1$	Much higher than single disk; comparable to RAID 2, 3, or 4	Similar to RAID 0 for read; lower than single disk for write	Similar to RAID 0 for read; generally lower than single disk for write
	6	Block-interleaved dual distributed parity	$N+2$	Highest of all listed alternatives	Similar to RAID 0 for read; lower than RAID 5 for write	Similar to RAID 0 for read; significantly lower than RAID 5 for write

4

RAID at CS dept. (S2007)

- File servers: group ja fs

RAID overhead 14% (1/7)

  - 6 RAID5 servers using 7 (+ 1 hot spare), each server total 1,6 TB
  - /group, /fs
  - */fs-0, /fs-1, /fs-2, /fs-3* (user directories dived to these)
  - Similar, but smaller disks: *courier.cs.helsinki.fi* (mailserver) - 0,9 TB

RAID overhead 20% (1/5)

- 4 RAID5 servers using 5 levyn (á 146 GB), á 550 GB
  - *backup* for backuping user files from fs
    - copy each /fs-i here and stream using 2 tape robots
    - SW RAID 5 in Linux kernel, since the robot controller and hardware RAID controller were not working together correctly
  - *db.cs.helsinki.fi* (database server)
  - *winserver.cs.helsinki.fi* (Windows 2003 Terminal Server)
  - *bodbacka.cs.helsinki.fi*

5

RAID at CS dept. (S2007)

- RAID 1 using 2 disks times 2 RAID-1 nameserver (Hydra)

RAID overhead 50% (1/2)
redundancy overhead 50%

- RAID 5 using 4 disks, total 0.9 TB (Bioinformatiikka)
- RAID 5 using 4 disks, total 0.45 TB (b-course)

RAID overhead 25% (1/4)

- 2 RAID 5 servers using 5 disks, each total 0.6 TB (Vera, Chuck)
  - Mainly for number crunching (32 GB), disk space is not crucial

6

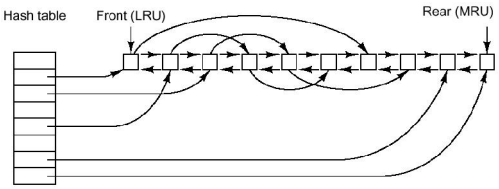
Buffering, disk cache Ch 11.7

- Buffer cache, block cache, disk cache
- OS uses a special data-area to store disk blocks that are read in memory
  - Use these to avoid unneeded read, read only if not in memory yet
- Locality principle is used also here
  - Files are quite often used in sequential order
  - Next reference to same (or the next) block
- Prefetch
  - Read already while processing the previous block
- Delayed write
  - Write first to the block in the memory (in buffer)
  - Write to the disk only when the block is full …
  - … or write the changed blocks (for example) every 30 secs to the disk

7

UNIX: block buffer

Tan01 6-27



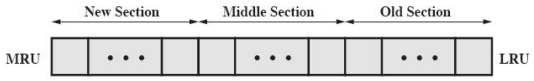
- Sentinel node (header node)
- device#, block#, links, Modified, Free
- Buffers on a separate dedicated area
- sentinel node contains link to the specific block
- Hash table for the search
- key: device#, block#

8

Replacement algorithms for buffers

- Limited size -> when no free areas, must replace existing one
  - Similar problem than in other buffering: TLB, cache, virtual memory
- If the block to be replaces has been changed, it must first be written to the disk before replacement
- Alternatives:
  - Least recently used
  - Least frequently used
  - Most recently used: FIFO or Three sections

Fig 11.9 (b) [Stal 05]



(b) Use of three sections

9

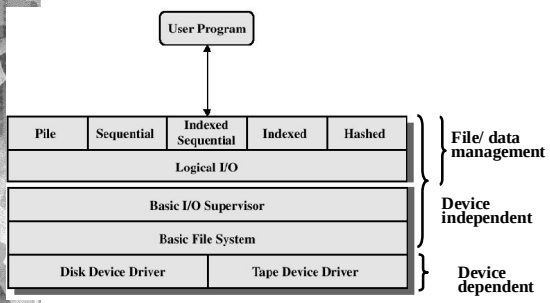
LUENTO 16

File system

Stallings, Luku 12.1-12.7

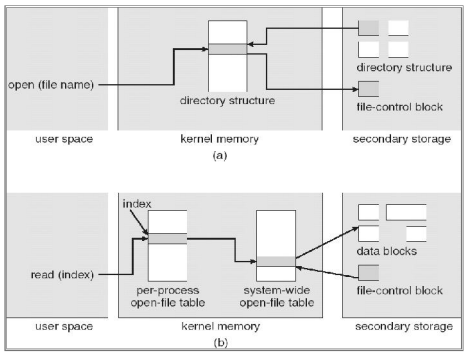
10

Architecture Kuva 12.1



11

Structures related to files [SGG07] Fig 11.3



12

File organisation

- Pile
- Sequential file
- Indexed sequential file
- Indexed file
- Direct or hased file

- Not usually task of OS, but OS might need to support different ways.

13

Table 12.1 Grades of Performance for Five Basic File Organizations [WIED87]

	Space		Update		Retrieval		
	Attributes		Record Size				
File Method	Variable	Fixed	Equal	Greater	Single record	Subset	Exhaustive
Pile	A	B	A	E	E	D	B
Sequential	F	A	D	F	F	D	A
Indexed sequential	F	B	B	D	B	D	B
Indexed	B	C	C	C	A	B	D
Hashed	F	B	B	F	B	F	E

A	= Excellent, well suited to this purpose	$\approx O(r)$
B	= Good	$\approx O(o \times r)$
C	= Adequate	$\approx O(r \log n)$
D	= Requires some extra effort	$\approx O(n)$
E	= Possible with extreme effort	$\approx O(r \times n)$
F	= Not reasonable for this purpose	$\approx O(n^{r-1})$

where
r = size of the result
o = number of records that overflow
n = number of records in file

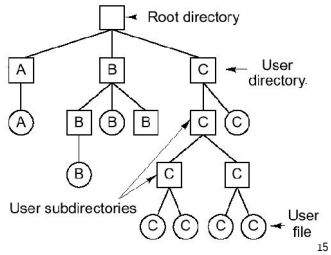
14

Directory

= File, that contains information about other files

- Only OS is allowed directly to access these files
  - All changes through system calls only

- Root directory, home directories
- Processes can create subdirectories
- Fixed location for root directory on disk



15

Table 12.2 Information Elements of a File Directory

Basic Information	
File Name	Name as chosen by creator (user or program). Must be unique within a specific directory.
File Type	For example: text, binary, load module, etc.
File Organization	For systems that support different organizations
Address Information	
Volume	Indicates device on which file is stored
Starting Address	Starting physical address on secondary storage (e.g., cylinder, track, and block number on disk)
Size Used	Current size of the file in bytes, words, or blocks
Size Allocated	The maximum size of the file
Access Control Information	
Owner	User who is assigned control of this file. The owner may be able to grant/deny access to other users and to change these privileges
Access Information	A simple version of this element would include the user's name and password for each authorized user.
Permitted Actions	Controls reading, writing, executing, transmitting over a network

16

Taulu 12.2. jatkuu

Usage Information	
Date Created	When file was first placed in directory
Identity of Creator	Usually but not necessarily the current owner
Date Last Read Access	Date of the last time a record was read
Identity of Last Reader	User who did the reading
Date Last Modified	Date of the last update, insertion, or deletion
Identity of Last Modifier	User who did the modifying
Date of Last Backup	Date of the last time the file was backed up on another storage medium
Current Usage	Information about current activity on the file, such as process or processes that have the file open, whether it is locked by a process, and whether the file has been updated in main memory but not yet on disk

17

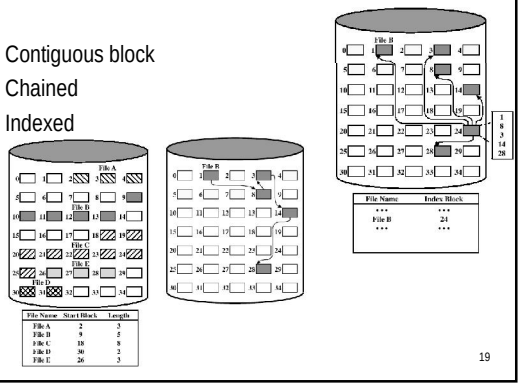
Shared files

- Usage rights collected to file attributes
- User grouping in (UNIX) u,g,o
- Rights (UNIX) r,w,x, -
- Directory rights (UNIX)
  - r right to list the directory content (read the directory element)
  - w right to remove a file from directory (write the directory element)
  - x right to use the directory name as part of the path name
- Some systems use different methods like access control / capability lists
- Access right is checked by OS only at the opening of file
  - Must have right to all parts of the path name

18

Allocation: Three alternatives

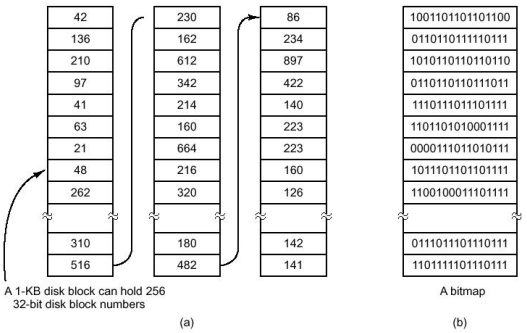
- Contiguous block
- Chained
- Indexed



19

Free space

Tan01 6-21



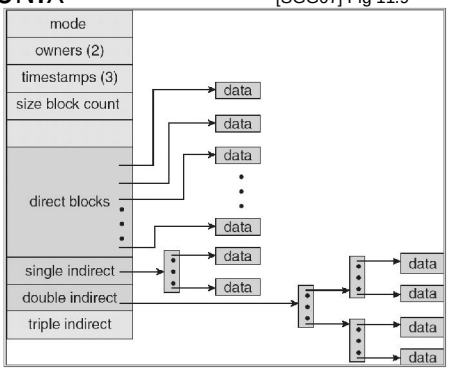
20

Esimerkki: Wanha perinteinen UNIX

21

UNIX

[SGG07] Fig 11.9



22

UNIX

Tan01 6-39

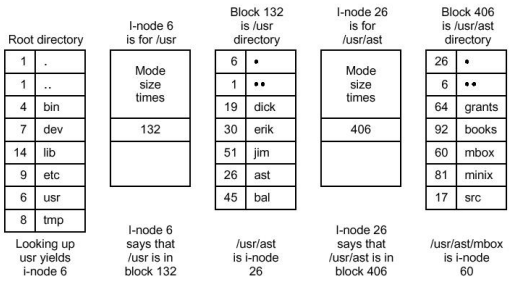


Fig. 6-39. The steps in looking up */usr/ast/mbox*.

23

UNIX

Tan01 10-31

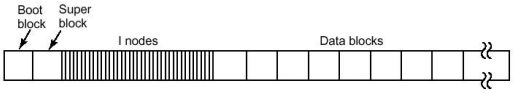


Fig. 10-31. Disk layout in classical UNIX systems.

- Location of boot-block is fixed by manufacturers, not unix specific
- i-nodes collected to one location, because they are small and block size is much larger. I-nodes quite often read to memory and used from there.
- Super block contains at least device#, partition size, start of the free block list, some numbers of free i-nodes

24

Rest of the slides copied from the "Finnish"

Linux siirräntä (kernel 2.6)

Deitel Ch 20.8 [DDC 04]

Deitel, Deitel & Choffnes: Operating Systems, 3rd ed., Pearson Prentice Hall, 2004.

25

Linux Device Drivers

- Loadable kernel modules
  - more than 50% kernel space?
- Devices in device special files in /dev
  - major id number (= device type)
    - determines driver
    - device drivers in /proc/devices
  - minor id number (device)
    - separates individual devices in same class
  - read/write/seek/ioctl to file invokes driver
  - device operations look like file ops
- Hot swappable devices ("plug-and-play" in Linux-land)
  - on enumerable bus positions (e.g., USB)
  - poll each position every now and then, find new device, identify it, and load kernel module for it

26

Linux Disk Scheduling

- Many algorithms provided
- Default algorithm: elevator variation
  - sort requests by track
  - try to merge with existing requests
  - LSTF – Least-Seek-Time-First

27

Linux Elevator Starvation Avoidance

- Problem: no guarantee of fast service, request can starve
  - busy writer can block lazy reader
- Solution #1: deadline scheduler — deadline vuorotus
  - each request has deadline (*read* 500 ms, *write* 5 s)
  - deadlines expire, expired requests will be serviced first
  - *read*/*write* FIFO queues to find expired requests quickly
  - group expired *read*s (and *write*s) together to minimize seeks
- Solution #2: anticipatory scheduler — ennakoiva vuorotus
  - synchronous (successive) *read*s happen usually once per timeslice
  - after each *read* wait for 6 ms (aver seek latency) for another *read* to arrive
    - if it arrives, service it first and avoid one seek
  - advantageous only if new reads happen more than 50% of the time
    - collect history data to decide if this method is useful
  - performance gain 5x-100x

28

Linux I/O Interrupts

- Each device has registered interrupt handler
  - interrupt handlers do not have context
  - interrupt handlers are not tasks (processes)
    - they can not be suspended or preempted
    - they can not cause exceptions
  - want to minimize time in interrupt handler
- top half — yläpuolisko
  - the real interrupt handler, fast, not a task, no context
  - schedules bottom half
- bottom half — alapuolisko
  - software interrupt handler, has context (e.g., device driver)
    - softirqs are suitable for SMP's, many concurrently
    - tasklets are suitable for mutex situations, one at a time
  - scheduled immediately after top half with high priority
    - if too many, all done one at a time with low priority

29

Linux Page Cache (for block devices)

- Same cache for VM pages and for memory mapped files
- If data is not in cache, place a request to a device request list
  - kernel sorts (may sort) requests by sector
  - kernel can optimize list for the device before submitting (part of) it
  - bio structure (Block I/O) maps memory to each request
- Kernel calls device driver with request list
  - device completes all requests in list
  - data transfer via kernel buffer cache
- HW RAID devices are given requests directly
- SW RAID implemented in kernel (included in std kernel)
- Linux Direct I/O does not use Page Cache (disk buffer)
  - direct copying from device to user space
    - no need to copy through kernel buffer cache
  - driver still suspends while waiting

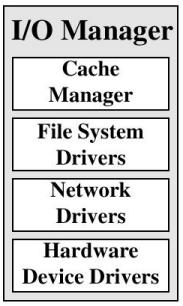
30

Windows 2000 I/O

Ch 11.10 [Stall 05]
Ch 11.6 [Tane 01]

31

W2K I/O-manager



(Fig 11.15 [Stal05])

- Device independent API for all devices
  - many different API's for all kinds of devices (device types)
- Dynamically loadable
- Cache
  - common for all file systems and networks
  - size varies dynamically
  - lazy *write* and *commit*
- Device drivers
  - access device registers via generic HAL interface
    - DLL for each platform

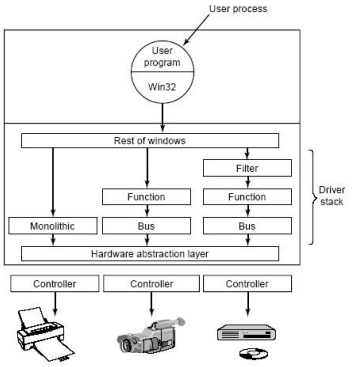
32

W2K Device Drivers

- Windows Driver Model
  - plug-and-play
  - re-entrant code, SMP supported
  - for each device, device object created in directory \??
  - W2000 and W98 support
- New device?
  - plug-and-play manager queries it for manufacturer and model
  - if recognized, load driver to memory from disk
  - if not recognized, ask for CD (or floppy), and then load it
- IRP (I/O Request Packet) for all I/O requests
- Drivers may be stacked
- Filter driver can do transformations for other drivers

Fig 11-30 [Tane01]

33



([Tane01]) Fig. 11-30. Windows 2000 allows drivers to be stacked.

34

W2K I/O

- File system
  - technically just a device with its own device driver
- SW RAID1
  - disk mirroring
    - shared device controller
  - disk duplexing
    - dedicated device controllers
- SW RAID 5
- Synchronous I/O
  - wait blocked until I/O completed
- Asynchronous I/O
  - send request and continue
  - later on, check that I/O completed and possibly wait
  - what can I do while waiting for I/O?

35

W2K Asynchronous I/O

- Send request and proceed
- Later on, check that I/O completed and possibly wait
  - signal via device kernel object
    - just one per device, so can wait for just one I/O request
  - signal via newly created event kernel object
    - any combination possible, because of many events
  - signal via thread APC queue (Asynchronous Procedure Call)
    - result of I/O-op to APC queue
    - APC executed later on (and signals thread requesting I/O?)
  - signal via specific I/O completion ports
    - fast
    - ready pool of server threads to serve the requests

36

W2K I/O Interrupts

(W XP, Ch 21.5 [DDC04])

- Fast response time by splitting interrupt handling to two parts
- Time critical in hardware interrupt service procedure
  - (same or lower level) interrupts disabled
  - no context
  - acknowledge interrupt, save interrupt state
  - invoke DPC or APC by triggering lowest level interrupts for them
- Rest in software interrupts DPC or APC (explained in next slides)
  - DPC (Deferred Procedure Call)
  - APC (Asynchronous Procedure Call)

Compare to Linux Top Half and Bottom Half!

37

W2K DPC

(W XP, Ch 21.5 [DDC04])

- DPC (Deferred Procedure Call)
- Software interrupts
- No own context, use interrupted thread context
  - for cases where context is not important
- Must not block in DPC
  - no context
- DPC queue at each processor
- Most of interrupt processing here

38

W2K APC

(W XP, Ch 21.5 [DDC04])

- APC (Asynchronous Procedure Call)
- Belongs to some thread, have context
  - invoke a procedure to be executed by someone else!
  - give thread something to do when it wakes up next time
  - usually I/O interrupt handler's not-so-time-critical part
- Special APC's have priority in execution order
  - before normal APC's
- Kernel mode APC (joka säikeellä jono)
  - software interrupts, generated by kernel mode components
  - executed only when owning thread is scheduled
  - all APC's done before thread can continue
- User mode APC (joka säikeellä jono)
  - threads can select when to execute APC's (if ever)
  - if thread never enters *alertable wait state*, then APC is never executed

39

W2K Cache Manager

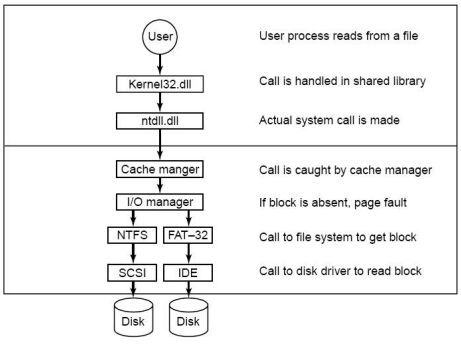
(Ch 11.9 [Tane01])
(Tiedostovälimuisti)

- One cache shared for all file systems
  - NTFS, FAT-32, FAT-16, CD-ROM, ...
  - sits on top of the file systems
  - based on logical files (file, offset)
    - not on physical files (partition, block)

Fig 11-45 [Tane01]

- All files are mapped to memory (in kernel address space)
  - access through virtual memory manager
    - read op: copy from kernel address space to user addr. space
  - access to disk buffer looks just like any memory access
  - page fault to cache manager handled just like any other page fault
    - cache manager does not know about it
    - cache manager does not even see physical memory
      - physical memory handled by VM in 256 KB chunks

40



[Tane01] Fig. 11-45. The path through the cache to the hardware.

41