

Levy I/O Linux ja W2000 levy I/O

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

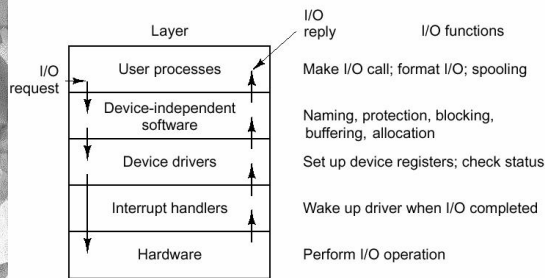
1

Sisältö

- RAID
- Lohkopuskurit (disk cache)
- Esimerkit: Linux, W2K

2

Siirännän hierarkia



(Fig 5-16 [Tane01])

3

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

Ch 11.6 [Stal 05]

4

RAID tavoitteet

- RAID (Redundant Array of Inexpensive/Independent Disks)
 - aluksi: ison levyn sijasta monta pienempää edullista levyä
 - nyt: monta kallista erillistä levyä tuo luotettavuutta
- Samanaikaisuus/rinnakkaisuus
 - tiedostot jaettu useammalle levyille
 - lohkon tavut jaettu useammalle levyille
 - hakuvarsin siirrot voivat tapahtua yhtä aikaa

5

RAID tavoitteet

- Koko levyjoukko näkyy KJ:lle yhtenä isona levynä
- Vikasietoisuus
 - vaikka levy hajoaa, sillä ollut tieto voidaan (toivottavasti) luoda uudelleen muiden levyjen perusteella
 - pariteetti, Mirror, Hamming, XOR ECC
 - ei RAID 0:ssa (onko siis edes RAID?)
 - "luku nolla tarkoittaa ei mitään"
 - RAID 0 antaa suorituskykyä, ei redundanssia

6

RAID tavoitteet

- Tavallisen levyohjaimen sijasta levyä ohjaa 'älykkäämpi' RAID-ohjain
 - ei tarvita muutoksia KJ:ään
 - huolehtii pariteetista
 - jos levy hajoaa, korjaa 'lennosta' bittijonoa, käyttäjä ei huomaa
 - levy irti, uusi tilalle, eheyttä se
- Myös ohjelmallisia toteutuksia (SW RAID)
 - yksi vikaantuva komponentti (RAID-ohjain) vähemmän
 - RAID-ohjain KJ:n ajurina
 - redundanssilaskenta CPU:lla erillisuorittimen (levyohjain) asemesta
 - sama funktionaalisuus kuin HW RAID-ohjaimella

7

RAID 0 (ei redundanssia, ei toisintoja)

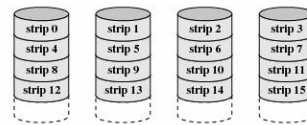


Fig 11.8a [Stal 05]

(a) RAID 0 (non-redundant)

- Monta levyä, fiksumin organisoituna
 - usean levyn optimointiratkaisu
- Ei redundanssia, Data hajautettu usealle levyille
- Etu: nopeus, samanaikaiset haut eri levyiltä

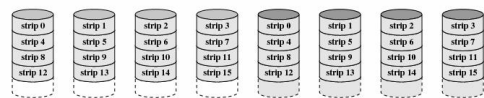
8

Stripe

- Stripe raita
 - looginen levy jaettu saman kokosiin strip-lohkoihin (levylohkon monikerta)
 - peräkkäiset strip-lohkot eri levyille
 - yhden tiedoston peräkkäiset sripit eri levyillä
 - samassa kohtaa eri levyillä olevat strip-lohkot muodostavat stripe raidan
 - tasapainottaa levykuormaa automaattisesti

9

RAID 1 (mirror, kahdennettu)



(b) RAID 1 (mirrored)

- Levyt tuplattu redundanssin vuoksi
 - mirror-levy
 - alkuaan tavalliset levylohkot, ei stripe'ja
- Levy rikkoutuu?
 - vaihda levy, päivitä tiedot "mirror"-levyltä

10

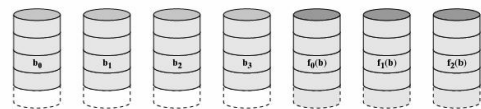
RAID 1 (mirror)

Fig 11.8b [Stal 05]

- Stripe-lohkot suorituskyvyn lisäämiseksi
 - oikeastaan "RAID 0-1" tällöin (kun stripe-lohkot mukana)
- Suorituskykyetu
 - luku kummalta tahansa levyllä (levyjoukolta, stripe'lta)
 - kummalla lyhyempi saantiaika?
 - kirjoitus molempiin (samanaikaisesti)
 - samanaikaisuutta eri levyille
- Tilakustannus?
 - 50% levytilasta (iso!)
- Usein käytössä, normaali optio levyohjaimessa ("mirror disk")

11

RAID 2 (Hamming)



(c) RAID 2 (redundancy through Hamming code)

- Kaikki levyt aina käytössä, strip-lohkon koko 1 bitti
- Hamming koodi
 - 4 data bittä -> 3 pariteettibittä, 4 levyä dataalle ja 3 pariteeteille
- Redundanssi
 - korjaa lennossa 1 bitin virheet, havaitsee 2 bitin virheet
- Levy rikkoontuu?
 - vaihda levy, laske bitit Hammingin avulla, alusta levy

12

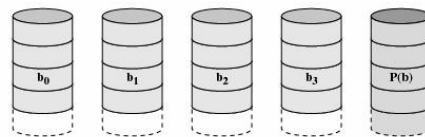
RAID 2 (Hamming)

- Ei suorituskykyetua – päinvastoin?
 - luku aina kaikilta levyiltä yhtä aikaa, ja sitten tarkistus
 - kirjoitus kaikille levyille
 - ei samanaikaisuutta eri levyitteissä
- Tilakustannus?
 - Hamming-koodin mukaan
 - 7 levyä: 3 pariteettilevyä eli 42%
 - 15 levyä: 4 pariteettilevyä eli 27%
 - yleensä liikaa

13

RAID 3 (pariteettibitti)

Fig 11.8d [Stal 05]



(d) RAID 3 (bit-interleaved parity)

- Pelkkä pariteettibitti, strip-lohkon koko 1 bitti
- Redundanssi
 - havaitsee 1 bitin virheet
 - korjaa lennossa
- Levy rikkoontuu?
 - vaihda levy, laske bitit pariteetin avulla, alusta levy

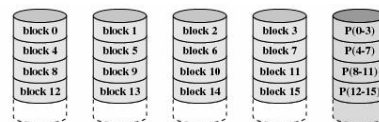
14

RAID 3 (pariteettibitti)

- Suorituskykyetu
 - luku ja kirjoitus aina kaikista levyistä (mutta rinnakkain)
 - jonotusaika ei vähene, mutta siirtonopeus kasvaa
 - ei samanaikaisuutta eri levyitteissä
- Tilakustannus?
 - aina 1 levy
- Ei paljon käytössä, koska huono suorituskyky

15

RAID 4 (pariteettilohko)



(e) RAID 4 (block-level parity)

- Pariteettilohko, strip-lohkon koko iso
- Redundanssi, jos levy rikkoontuu
 - korjaa lennossa 1 bitin virheet
 - vaihda levy, laske lohkot pariteetin avulla, alusta levy

Fig 11.8e [Stal 05]

16

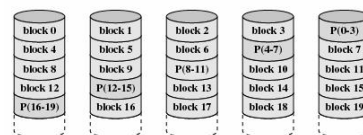
RAID 4 (pariteettilohko)

- Suorituskykyetu
 - luku rinnakkain eri levyistä, levyt toimivat itsenäisesti
 - kirjoituksessa yhteen lohkoon muut saman stripe'n lohkot pitää ensin lukea pariteettilohkonlaskemista varten
 - tai sitten lue ensin vanha lohko ja laske siitä
 - kirjoituksessa pitää aina kirjoittaa myös pariteettilohko
 - jonotusaika vähenee, siirtonopeus kasvaa
 - samanaikaisuutta eri levyitteissä
- Tilakustannus?
 - aina 1 levy
- Ei paljon käytössä, koska pariteettilevy on pullonkaula

17

RAID 5 (hajautettu pariteettilohko)

Fig 11.8f [Stal 05]



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

- Kuten RAID 4, mutta pariteettilohko vuorotellen eri levyille
 - kirjoittamisen yhteydessä ei enää pariteettilevyn pullonkaulaa
- Yleisesti käytössä oleva

18

RAID 6 (2 haj. pariteettilohkoa)

(g) RAID 6 (dual redundancy)

Fig 11.6g [Stal 05]

- kuten RAID 5, mutta tarkistuslohkot laskettu kahdella eri menetelmällä
- Redundanssi
 - korjaa lennossa 2 bitin virheet
- 1 tai 2 levyä rikkoontuu?
 - vaihdta levyt, laske lohkot eri menetelmillä, alusta levyt

19

RAID 6 ja muut

- Ei niin standardoitu kuin RAID 0, ... RAID 5
- RAID 6 (2 hajautettua pariteettilohkoa)
 - tilakustannus?
 - aina kaksi tarkistuslohkolevyä
- On muitakin...
 - RAID 0+1, RAID 0+3, RAID 0+5, RAID 1+5, RAID 10, RAID 50, RAID 51, RAID 53, ...
 - RAID 7, Storage Computer'in patentti
 - nopeampi kuin RAID 3 tai RAID 5, luotettava, kallis
 - yksi pariteettilevy, levyohjaimessa suuri cache

20

RAID yhteenveto

Tbl 11.4 [Stal 05]

21

RAID ja TKTL (K2006)

- Tiedostopalvelimet group ja fs
 - 2 kpl 7 (+ 1 hot spare) levyn RAID 5 serveriä, ä 2 TB
 - /group, /fs (mm. verkkopalvelua)
- 8 kpl 5 levyn (ä 146 GB) RAID 5 servereitä, ä 550 GB
 - /fs-0, /fs-1, /fs-2, /fs-3 (tavalliset tiedostopalvelimet)
 - backup varmuuskopioita varten
 - kopioi /fs-i tänne ja stream'aa 2 nauharobotille
 - SW RAID 5 Linuxin ytimessä, koska nauharobotit eivät sopineet yhteen RAID-ohjaimen kanssa
 - db.cs.helsinki.fi (tietokantapalvelin)
 - winsrvr.cs.helsinki.fi (Windows 2003 Terminal Server)
 - courier.cs.helsinki.fi (postipalvelin)

RAID kust. 14% (1/7)

RAID kust: 20% (1/5)

22

RAID ja TKTL (K2006)

- 2 krt 2 levyn RAID-1 nimipalvelin (Hydra)
 - RAID kust: 50% (1/2)
 - redund. kust 50%
- 4 levyn RAID 5 serveri, ä 0.9 TB (Bioinformatiikka)
- 4 levyn RAID 5 serveri, ä 0.45 TB (CoSCO)
 - RAID kust: 25% (1/4)
- 2 kpl 5 levyn RAID 5 serveriä, ä 0.6 TB (Vera, Chuck)
 - Varsinaisesti laskentapalvelimia, levytila toissijaista

23

Lohkopuskurit

Ch 11.7 [Stal 05]

24

Lohkopuskurit, levypuskurit

- myös nimellä levyvälimuisti (**disk cache**)
- Buffer cache, block cache, disk cache
- KJ:n data-alueella oleva puskuri muistiinluettuja levylohkoja varten
 - jos viitattu lohko muistissa, ei levynoutoa
 - jokainen I/O-pyyntö ei aiheuta levyliikennettä
 - useimmat eivät
- Tasaa erot kerralla käsiteltävän yksikön kanssa
 - ohjelma lukee/kirjoittaa tavuja
 - levyohjain lukee/kirjoittaa lohkoja

Ranskan kielen sana "cache" tarkoittaa piilottamista

25

Lohkopuskurit

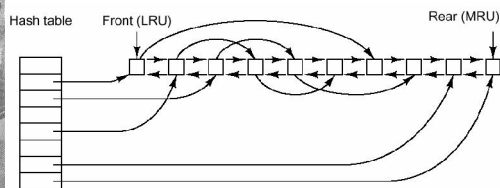
- Paikallisuusperiaate pätee myös levyä käytettäessä
 - tiedostoa käydään läpi yleensä peräkkäisjärjestyksessä
 - seuraava viite todennäköisesti samaan lohkoon
- Ennaltanouto
 - kun tiedosto avataan, hae ens. lohko heti lohkopuskuriin
 - seuraavan nouto, heti kun edellistä käsitellään
 - (usean) seuraavan nouto samalla kertaa
- Viivästetty kirjoitus
 - talleta ensin levypuskuriin
 - kirjoita vasta täysi puskuri levyille ...
 - ... tai kirjoita muuttuneet esim. 30 sek välein levyille

vrt. virt.muistin
cleanup

26

UNIX: Lohkopuskuri

Tan01 6-27



- Tunnusolmu
 - laite#, lohko#, linkkejä, Modified, Free
- Puskurit kokonaisina erillisellä alueella
 - tunnusolmussa viite varsinaiseen puskuuriin
- Hajautustaulu etsinnän nopeuttamiseksi
 - avaimena laite#, lohko#

27

Lohkopuskurin poistoalgoritmi

- Tilaa varattu rajallisesti
 - UNIX: esim 100 ... 1000 puskuria
 - Linux: koko vapaana oleva muisti (usein 50% muistista)
- Kun ei enää tilaa uusille lohkoille, joku lohko poistettava puskurista
- Samat ongelmat kaikessa puskuroidinnissa
 - TLB: mikä alkio korvataan?
 - välimuisti: mikä muistilohko korvataan?
 - levypuskuri: mikä levylohko korvataan?
 - virtuaalimuisti: mikä sivutila / segmentti korvataan?
- Jos poistettava lohko on muuttunut, se täytyy kirjoittaa takaisin levyille

28

LRU: Least Recently Used

- Poista lohko, johon viittaamisesta kulunut kauimmin aikaa
- Loogisesti ajatellen:
 - levyvälimuisti on jono lohkoja
 - viimeksi viitattu lohko jonon viimeisenä
 - päivitys joka viittauskerralla?
 - poista jonon ensimmäinen lohko
- Toteutus: jonossa osoittimia lohkoihin
 - lohkoja ei tarvitse järjestellä
 - tietyn lohkon etsinnän tehokkuus?
- Poistettava voi silti olla tarpeellinen - sitä käyttävä prosessi sattui olemaan Blocked-tilassa
 - seuraus: pitää lukea levyltä pian uudelleen

29

LFU: Least Frequently Used

- Poista lohko, johon vähiten viittauksia
- Tarvitaan lohko kohtainen viitelaskuri
 - kasvata aina, kun lohkoon viitataan
 - laskurin nollaus aika-ajoin
- Lohko, jonka viitelaskuri on suuri, saattaa silti olla tarpeeton
 - prosessin vaihe, jossa lohkoa tarvittiin, on jo mennyt

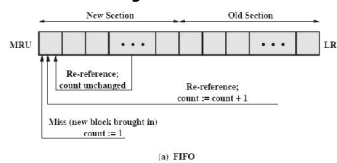
30

Most Recently Used – MRU FIFO (Frequency Based Replacement)

- Yrittää huomioida molemmat edelliset ideat (LRU + LFU)
- Most Recently Used – MRU FIFO
 - jos uusi viite äskettäin viitattuun lohkoon (jonon loppuosassa)
 - lohko jonon perään
 - älä kasvata viitelaskuria
 - useat peräkkäiset viitteet kasvattavat laskuria vain kerran
 - jos viite kauan sitten viitattuun lohkoon (jonon alkuosassa)
 - lohko jonon perään
 - kasvata viitelaskuria
 - poista loppuosasta (old section) lohko, jonka viitelaskuri pienin
 - jos useilla sama arvo, poista takimmaisempi (LRU)

31

Most Recently Used – MRU FIFO



- Most Recently Used – MRU FIFO
 - ajatus hyvä 10, mutta vain vähän parannusta
 - kun lohkoon viitataan se säilyy alkuosassa
 - kun viittaukset loppuvat, lohko putoaa loppuosaan
 - viimeksi loppuosaan joutuneella loholla pieni viitelaskurin arvo
 - vaikka käytetty viimeksi, perällä olevista joutuu helpoimmin poiston kohteeksi, ellei siihen viitata pian uudestaan

Fig 11.9 (a) [Stal 05]

32

Most Recently Used – MRU Three Sections

- Most Recently Used – MRU Three Sections
 - parannus: jaa jono kolmeen osaan
 - poistot aina viimeisestä osasta
 - etuosasta pudonneelle jää aikaa vanheta
 - tulos: parempi algoritmi kuin LRU tai LFU

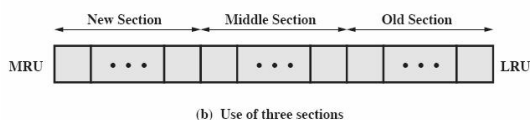


Fig 11.9 (b) [Stal 05]

33

Linux siirräntä (kernel 2.6)

Deitel Ch 20.8 [DDC 04]

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

34

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

35

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

36

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 *reads* (and *writes*) together to minimize seeks
- Solution #2: anticipatory scheduler **ennakoiva vuorotus**
 - synchronous (successive) *reads* 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 more than 50% of the time
 - collect history data to decide if this method is used
 - performance gain 5x-100x

37

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 a time with low priority

38

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

39

Windows 2000 I/O

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

40

W2K I/O-manager

I/O Manager

Cache
Manager

File System
Drivers

Network
Drivers

Hardware
Device Drivers

- 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

(Fig 11.15 [Stal05])

41

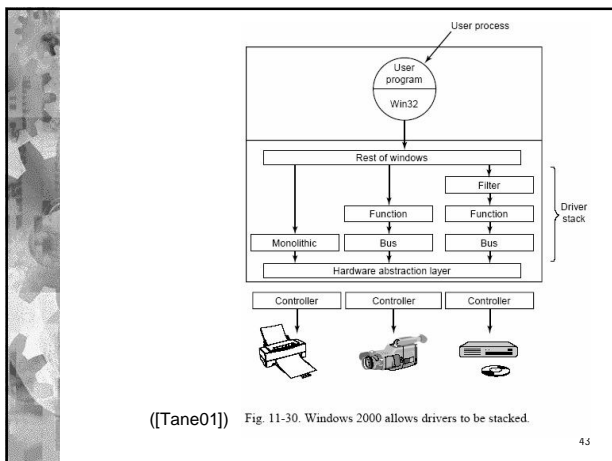
W2K Device Drivers

- 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 **Fig 11-30 [Tane01]**
- Drivers may be stacked
- Filter driver can do transformations for other drivers

42

Syksy 2006 / Luento 15

15-7



- ### 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?
- 44

- ### 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 ports
- 45

- ### 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!**
- 46

- ### 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
- 47

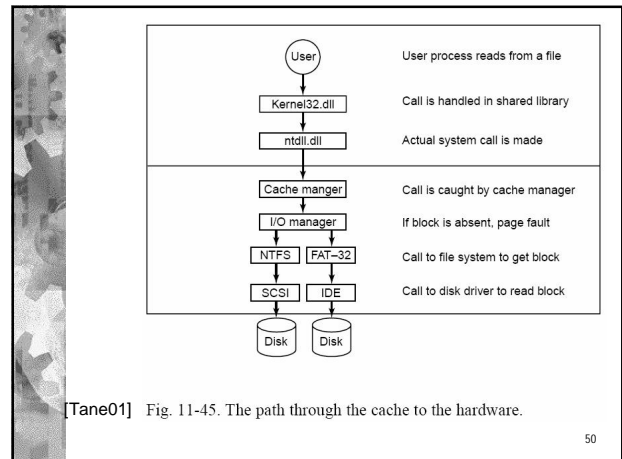
- ### W2K APC (W XP, Ch 21.5 [DDC04])
- APC (Asynchronous Procedure Call)
 - Belongs to some thread, have context
 - invoke a procedure to be executed by someone else!
 - give thread something to do when he wakes up next time
 - usually I/O interrupt handler's not-so-time-critical part
 - Special APC's 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
- 48

(Ch 11.9 [Tane01])

W2K Cache Manager (Tiedostovälimuisti)

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

49



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