LUENTO 9

Käyttöjärjestelmät II

Levy I/O
Linux ja W2000 levy I/O

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

Mitä KJ-I:ssä / KJ-II:ssa?

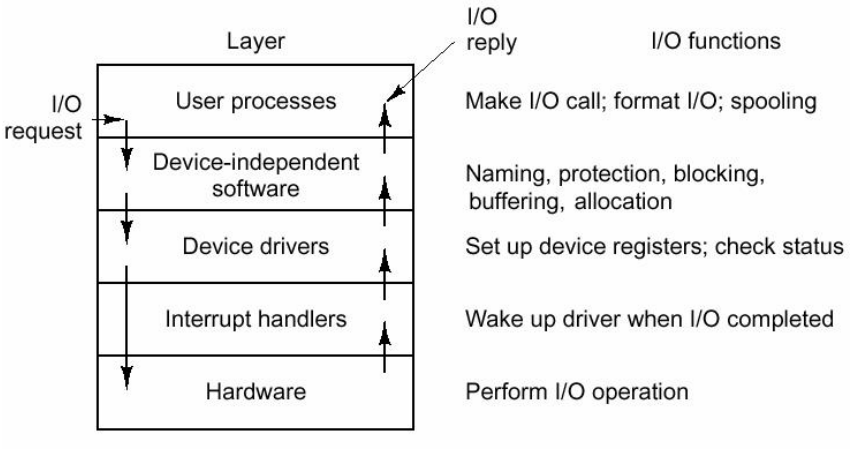
KJ-I

- I/O-laitteista
- I/O:n organisointi
- KJ:n suunnittelusta
- Puskurointi

Seuraavaksi KJ-II:ssa, tämä luento

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

Siirrännän hierarkia



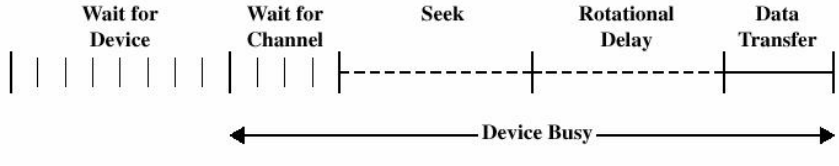
(Fig 5-16 [Tane01])

Käyttöjärjestelmät II

Levypyyntöjen järjestely

Ch 11.5 [Stal 05]

Levyhaku



(Fig 11.6 [Stal05])

- **Laitteen vapautumisen odotus**
  - ohjain käsittelee yhden pyynnön kerrallaan
- **Siirtokanavan odotus**
  - jos useita levyjä samassa väylässä
- **Hakuvarren siirto-aika** (seek time)
  - hakuvarsi oikealle uralle
- **Pyörähdysviive** (rotational delay/latency)
  - odota, että oikea sektori pyörähtää kohdalle
- **Siirtoaika** (transfer time)
  - yhden lohkon kirjoittamiseen/lukemiseen kuluva aika

saantiaika (access time)

?

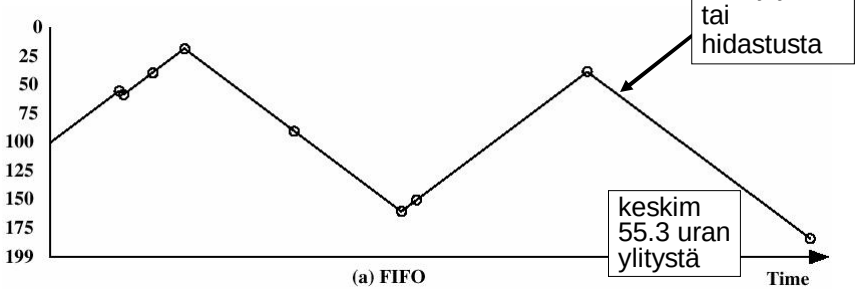
Algoritmeja

- **Hakuvarren siirtoaika pisin**
  - kannattaa minimoida siirrot
- **Random? FIFO? PRI? LIFO?**
  - huonoja, eivät huomioi hakuvarren nykyistä positiota
- **Ota huomioon hakuvarren sijainti**
  - SSTF
  - SCAN
  - C-SCAN
  - N-step-SCAN ja FSCAN
- **Esimerkeissä: 200 uraa, hakuvarsi uralla 100, hae urilta 55, 58, 39, 18, 90, 160, 150, 38, 184**

Nyt 100; hae 55, 58, 39, 18, 90, 160, 150, 38, 184

First-in-first-out, FIFO

- **Käsittele saapumisjärjestyksessä**
- **Tasapuolinen kaikille prosesseille**
- **Prosessin levypyynnöt usein toistensa lähialueilta**
  - FIFO ei ihan yhtä huono kuin Random!
- **Ei hyvä, koska ei ota huomioon levyn tilaa**



Nyt 100; hae 55, 58, 39, 18, 90, 160, 150, 38, 184

(Fig 11.7 [Stal 05])

Prioriteetti, PRI

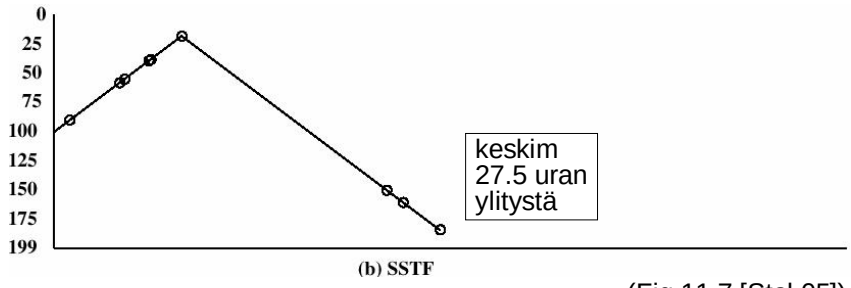
- **Käytä prosessin suoritin prioriteettia myös levyprioriteettina**
- **Prosessin prioriteetti määräytyy muiden tekijöiden (suoritin!) kuin levyhakujen perusteella**
- **Lyhyillä erätöillä usein suuri (suoritin) prioriteetti**
  - minimoi läpimenoaikaa
- **Interaktiivisilla hyvä (suoritin) prioriteetti**
  - minimoi vastausaikaa
- **Ei hyvä, koska ei ota huomioon levyn tilaa**

LIFO, Last-in-first-out

- **Palvele viimeksi tullut pyyntö ennen muita**
  - anna laite aina aktiivisimmalle prosessille
    - idea: vähentää hakuvarren siirtotarvetta, erityisesti peräkkäistiedostoja käsiteltäessä (paikallisuusilmiö)
- **Nälkiintymisvaara**
  - jos paljon I/O-sidonnaisia prosesseja, vanha pyyntö voi jäädä jalkoihin
    - *non-blocking write* – aina läpi
    - *blocking read* – voi odottaa kauan
- **Ei hyvä, koska ei ota huomioon levyn tilaa**

SSTF, Shortest Service Time First

- **Palvele pyyntö, jossa selvitään lyhyimmällä hakuvarren siirrolla**
- **Esiintyy myös nimellä Shortest Seek First (SSF)**
- **Vanhat voi jäädä jalkoihin, kun paljon uusia töitä**
  - I/O sid. työ (korkea CPU prioriteetti)?
    - lue uralta 25 ⟶ käytä CPU:ta ⟶ lue uralta 25 ⟶ …



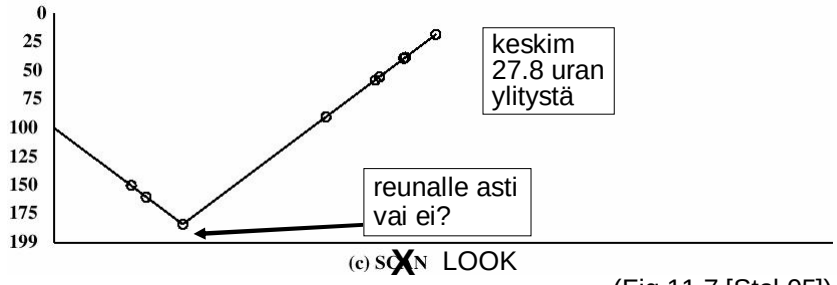
keskim 27.5 uran ylitystä

(Fig 11.7 [Stal 05])

Nyt 100; hae 55, 58, 39, 18, 90, 160, 150, 38, 184

SCAN

- Siirrä hakuvartta samaan suuntaan kunnes reuna vastaan
- Vaihda sen jälkeen hakuvarren suunta, ja palvele matkan varrelle osuvat pyynnöt
- Suosii **keskiurille** osuvia pyyntöjä ja uusia töitä
- Myös nimellä **elevator** (hissi)
- **LOOK**-versio: käänny takaisin heti, jos edessä ei työtä
  - ei mennä reunalle asti



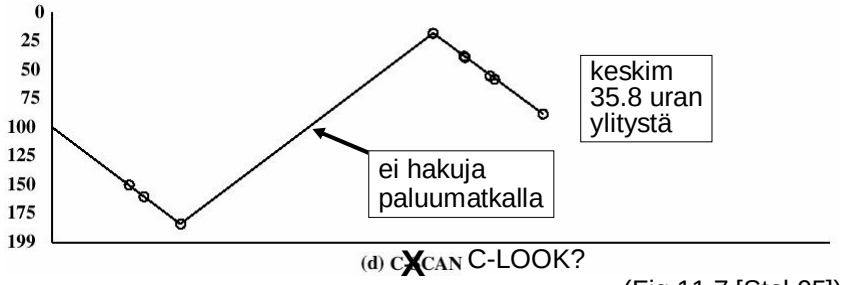
(c) SCAN X LOOK

Nyt 100; hae 55, 58, 39, 18, 90, 160, 150, 38, 184

(Fig 11.7 [Stal 05])

Circular-Scan, C-SCAN

- **Siirrä hakuvartta käsittelyn aikana aina samaan suuntaan**
- **Kun ko. suunnassa ei ole enää palveltavia, palauta hakuvarsi takaisin ja aloita uudelleen**
- **Hukka-aika varren palauttamiseen?**
- **Kaikilla urilla sama palvelu**
- **C-LOOK versio: ei mennä reunalle asti, jos ei tarvitse**



(d) CSCAN X C-LOOK?

Nyt 100; hae 55, 58, 39, 18, 90, 160, 150, 38, 184

(Fig 11.7 [Stal 05])

FSCAN

- **Ongelma: paljon uusia töitä samalle uralle**
  - vanhat työt odottavat ja odottavat…
- **Ratkaisu: Kaksi jonoa (S ja Q, eli Service ja Queue)**
  - palvele S-jonoa (SCAN-algoritmilla)
    - näiden palvelun aikana kaikki uudet työt jäävät jonoon
  - kerää Q-jonoon sillä aikaa tulevat pyynnöt
  - kerralla palvellaan kaikki edellisen S-jonon käsittelyn aikana tulleet työt
- **Vasteajan varianssi pienempi kuin SCAN'illa**

KJ-II K2006 / Auvo Häkkinen - Teemu Kerola 6.4.2006 13

N-step-SCAN

- **Ongelma: FSCAN:issa paljon uusia töitä samalle uralle**
  - vanhat työt odottavat ja odottavat…
  - S-jono hyvin pitkä, jolloin sen alussa olevat työt voivat joutua odottamaan paljon, kunnes oma vuoro tulee
- **Ratkaisu: kaksi jonoa (S ja Q, Service ja Queue)**
  - palvele S-jonoa (SCAN algoritmilla)
    - näiden palvelun aikana kaikki uudet työt jäävät jonoon
  - kerää Q-jonon loppuun sillä aikaa tulevat pyynnöt
  - ota uuteen S-jonoon ***n* ensimmäistä jonottajaa** Q-jonosta
    - tai kaikki, jos vähemmän kuin *n*
  - kerralla palvellaan korkeintaan *n* työtä
    - n = 1 $\longrightarrow$ FIFO
    - n = ∞ $\longrightarrow$ SCAN
  - miksi nimi **N-step**-SCAN?

KJ-II K2006 / Auvo Häkkinen - Teemu Kerola 6.4.2006 14

Algoritmien vertailu

Tbl. 11.2 [Stal 05]

Seek Time	80GB	400GB
Average (1/3)	12	4.2
Track-to-track	2.5	2.0
Full stroke	23	21 ms

- **Hyvyyden mitta?**
  - läpikäytyjen urien lkm?
- **Parempi hyvyyden mitta**
  - hakuvarren siirtoaika yhteensä?
    - ei ole lineaarinen ylitettävien urien suhteen
  - kokonaisaika? pyörähdysviive? vasteaika?
- **Yhteenveto**

Tbl. 11.3 [Stal 05]

Pyörähdysviive mukaan - realismia

(ks. [DDC 01])

- **SLTF – Shortest Latency Time First**
  - minimoi **pyörähdysviive**
  - oma jono joka sektorille (uralle), ”*sector queueing*”
- **SPTF – Shortest Positioning Time First**
  - minimoi **hakuvarren siirtoaika + pyörähdysviive**
  - etusija keskiurille, sisä- ja ulkoreunoja sorsitaan
- **SATF - Shortest Access Time First**
  - minimoi **hakuvarren siirtoaika + pyörähdysviive + tiedonsiirtoaika** (access time = tiedon sijainnin etsintä + siirtoaika)
  - etusija keskiurille, sisä- ja ulkoreunoja sorsitaan
  - etusija pienillä töillä

Rot. delay	80GB	400GB
Rot. speed	4800	7200 rpm
Aver. rot. delay	6.25	4.2 ms

Loogiset sektorit

- **Peräkkäiset sektorinumerot eivät välttämättä tarkoita, että sektorit olisivat fyysisesti peräkkäin**
  - ylimääräiset sektorit virheenkorjausta ja –havaitsemista varten
  - viallinen sektori 123? Otetaan tilalle varasektori 9876, jonka looginen osoite on silti 123
  - eivät näy ulospäin lainkaan, levyohjaimen sisäistä tietoa
    - laiteajuri **ei voi** ottaa huomioon!
- **Laiteajurin (kernelin) tekemä optimointi voi perustua väärään tietoon**
  - perustuu oletukseen, että looginen osoite olisi sama kuin fyysinen levyosoite
- **Joissakin levyissä voidaan kysellä todellista sektorien sijaintia**

Käyttöjärjestelmät II

RAID

Redundant Array of Independent Disks

Ch 11.6 [Stal 05]

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 levylle
  - lohkon tavut jaettu useammalle levylle
  - hakuvarsien siirrot voivat tapahtua yhtäaikaa
- **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

RAID tavoitteet

- **Koko levyjoukko näkyy KJ:lle yhtenä isona levynä**
- **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, eheytä se
- **Myös ohjelmallisia toteutuksia (SW RAID)**
  - yksi vikaantuva komponentti (RAID-ohjain) vähemmän
  - RAID-ohjain KJ:n ajurina
  - redundanssisuuslaskenta CPU:lla erillissuorittimen (levyohjain) asemesta
  - sama funktionaalisuus kuin HW RAID-ohjaimella

RAID 0 (data hajautettu usealle levylle)

- **Monta levyä, fiksusti organisoituna**
  - usean levyn optimointiratkaisu
- **Ei redundanssia**
- **Data hajautettu usealle levylle**
- **Etu: nopeus, samanaikaiset haut eri levyiltä**
- **Stripe raita**
  - looginen levy jaettu saman kokoisiin **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

Fig 11.10 [Stal 01]

Fig 11.8a [Stal 05]

RAID 1 (mirror)

Fig 11.8b [Stal 05]

- **Levyt tuplattu redundanssin vuoksi**
  - mirror-levy
  - alkuaan tavalliset levylohkot, ei stripe'ja
- **Stripe-lohkot suorituskyvyn lisäämiseksi**
  - oikeastaan "**RAID 0-1"** tällöin (kun stripe-lohkot mukana)
- **Suorituskykyetu**
  - luku kummalta tahansa levyltä (levyjoukolta, stripe'lta)
    - kummalla lyhyempi saantiaika?
  - kirjoitus molempiin (samanaikaisesti)
  - samanaikaisuutta eri levyviitteissä
- **Levy rikkoutuu?**
  - vaihda levy, päivitä tiedot "mirror"-levyltä
- **Tilakustannus?**
  - 50% levytilasta (iso!)
- **Usein käytössä, normaali optio levyohjaimessa ("mirror disk")**

RAID 2 (Hamming)

Fig 11.8c [Stal 05]

- **Kaikki levyt aina käytössä, strip-lohkon koko 1 bitti**
- **Hamming koodi**
  - 4 data bittiä -> 3 pariteettibittiä, 4 levyä datalle ja 3 pariteeteille
- **Ei suorituskykyetua – päinvastoin?**
  - luku aina kaikilta levyiltä yhtä aikaa, ja sitten tarkistus
  - kirjoitus kaikille levyille
  - ei samanaikaisuutta eri levyviitteissä
- **Redundanssi**
  - korjaa lennossa 1 bitin virheet, havaitsee 2 bitin virheet
- **Levy rikkoontuu?**
  - vaihda levy, laske bitit Hammingin avulla, alusta levy
- **Tilakustannus?**
  - Hamming-koodin mukaan
  - 7 levyä: 3 pariteettilevyä eli 42%
  - 15 levyä: 4 pariteettilevyä eli 27%
  - **liikaa**.

RAID 3 (pariteettibitti)

Fig 11.8d [Stal 05]

- **Pelkkä pariteettibitti, strip-lohkon koko 1 bitti**
- **Suorituskykyetu**
  - luku ja kirjoitus aina kaikista levyistä (mutta rinnakkain)
  - jonotusaika ei vähene, mutta siirtonopeus kasvaa
  - ei samanaikaisuutta eri levyviitteissä
- **Redundanssi**
  - havaitsee 1 bitin virheet
  - korjaa lennossa
- **Levy rikkoontuu?**
  - vaihda levy, laske bitit pariteetin avulla, alusta levy
- **Tilakustannus?**
  - aina 1 levy
- **Ei paljon käytössä, koska huono suorituskyky**

RAID 4 (pariteettilohko)

Fig 11.8e [Stal 05]

- **Pariteettilohko, strip-lohkon koko iso**
- **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 levyviitteissä
- **Redundanssi, jos levy rikkoontuu**
  - korjaa lennossa 1 bitin virheet
  - vaihda levy, laske lohkot pariteetin avulla, alusta levy
- **Tilakustannus?**
  - aina 1 levy
- **Ei paljon käytössä, koska pariteettilevy on pullonkaula**

RAID 5 (hajautettu pariteettilohko)

Fig 11.8f [Stal 05]

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

RAID 6 ja muut

- **Ei niin standardoitu kuin RAID 0, … RAID 5**
- **RAID 6** (2 hajautettua pariteettilohkoa)
  - kuten RAID 5, mutta tarkistuslohkot laskettu kahdella eri menetelmällä
  - redundanssi
    - korjaa lennossa 2 bitin virheet
  - 1 tai 2 levyä rikkoontuu?
    - vaihda levyt, laske lohkot eri menetelmillä, alusta levyt
  - 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

Fig 11.6g [Stal 05]

Tbl 11.4 [Stal 05]

RAID ja TKTL (K2005)

- **2 kpl 7 (+ 1 hot spare) levyn RAID 5 serveriä, á 2 TB**
  - **/group**, **/fs** (mm. verkkopalvelua)
- **8 kpl 5 levyn (á 146 GB) RAID 5 servereitä, á 550 GB**
  - ***/fs-0, /fs-1, /fs-2, /fs-3*** (tavalliset tiedostopalvelimet)
  - ***backup*** varmuuskopiota 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)
  - ***winserver.cs.helsinki.fi*** (Windows 2003 Terminal Server)
  - ***courier.cs.helsinki.fi*** (postipalvelin)
- **2 krt 2 levyn RAID-1 nimipalvelin (Hydra)**
- **4 levyn RAID 5 serveri, á 0.9 TB (Bioinformatiikka)**
- **4 levyn RAID 5 serveri, á 0.45 TB (CoSCO)**
- **2 kpl 5 levyn RAID 5 serveriä, á 0.6 TB (Vera, Chuck)**

RAID kust. 14% (1/7)

RAID kust: 20% (1/5)

RAID kust: 50% (1/2)
redund. kust 50%

RAID kust: 25% (1/4)

Käyttöjärjestelmät II

Lohkopuskurit

Ch 11.7 [Stal 05]

Lohkopuskurit, levypuskurit

- **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 koossa**
  - ohjelma lukee/kirjoittaa tavuja
  - levyohjain lukee/kirjoittaa lohkoja

Ranskan kielen sana ”*cacher*” tarkoittaa piilottamista

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 levylle …
  - … tai kirjoita muuttuneet esim. 30 sek välein levylle

vrt. virt.muistin cleanup

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 puskuroinnissa**
  - 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 levylle**

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

LFU: Least Frequently Used

- **Poista lohko, johon vähiten viittauksia**
- **Tarvitaan lohkokohtainen 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

Most Recently Used – MRU FIFO

- **Yrittää huomioida molemmat edelliset ideat (LRU + LFU)**
- **Most Recently Used – MRU FIFO** Fig 11.9 (a) [Stal 05]
  - 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)

Most Recently Used – MRU Three Sections

- **Most Recently Used – MRU FIFO** Fig 11.9 (a) [Stal 05]
  - ajatus hyvä 10, mutta vain vähän parannusta
  - kun lohkoon viitataan se säilyy alkuosassa
  - kun viittaukset loppuvat, lohko putoaa loppuosaan
  - viimeksi loppuosaan joutuneella lohkolla pieni viitelaskurin arvo
    - vaikka käytetty viimeksi, perällä olevista joutuu helpoimmin poiston kohteeksi, ellei siihen viitata pian uudestaan
- **Most Recently Used – MRU Three Sections** Fig 11.9 (b) [Stal 05]
  - parannus: jaa jono kolmeen osaan
    - poistot aina viimeisestä osasta
    - etuosasta pudonneelle jää aikaa vanheta
  - tulos: parempi algoritmi kuin LRU tai LFU

Käyttöjärjestelmät **II**

Linux siirräntä (kernel 2.6)

Deitel Ch 20.8 [DDC 04]

(verkossa aikataulusivulla)

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

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

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
  - advantegous only if new reads more than 50% of the time
    - collect history data to decide if this method is used
  - performance gain 5x-100x

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)
    - **softirq**s are suitable for SMP's, many concurrently
    - **tasklet**s 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

Linux Page Cache (for block devices)

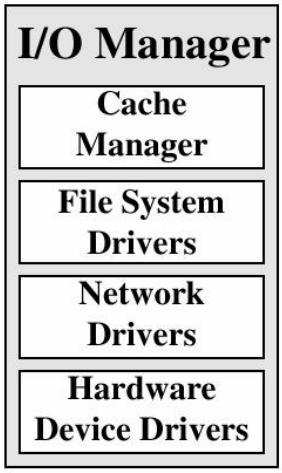
- **Same cache for VM pages and for memory mapped files**
- **If data is not in cache, place a request to a device request list**
  - kernel sorts (may sort) requests by sector
  - kernel can optimize list for the device before submitting (part of) it
  - **bio structure** (Block I/O) maps memory to 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

Operating Systems II

Windows 2000 I/O

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

W2K I/O-manager



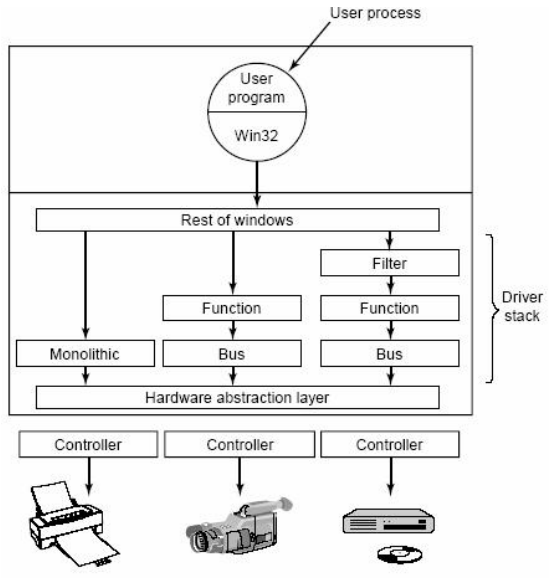
(Fig 11.15 [Stal05])

- **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

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]



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

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?

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

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**
  - DPC (Deferred Procedure Call)
  - APC (Asynchronous Procedure Call)

(explained in next slides)

Compare to Linux Top Half and Bottom Half!

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**
- **DPC queue at each processor**
- **Most of interrupt processing here**

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
    - owning thread wakes needs to be in ***alertable wait state***
  - 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

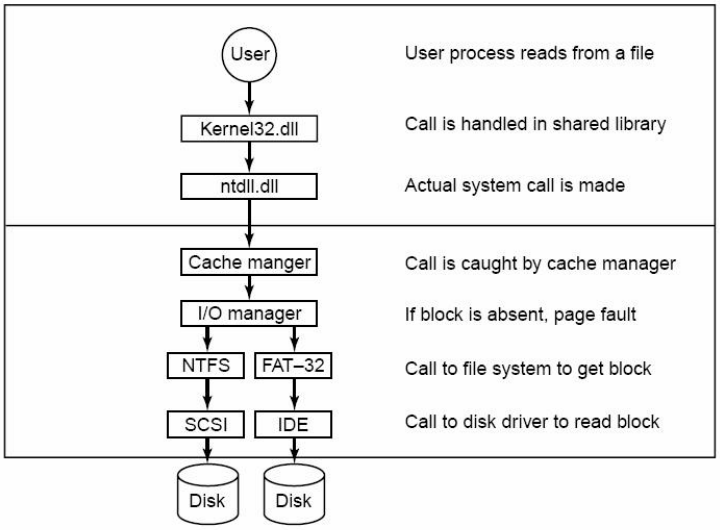
W2K Cache Manager

(Ch 11.9 [Tane01])

(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 sp.
  - 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

Fig 11-45 [Tane01]



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

Kertauskysymyksiä

- **Siirrännän hierarkia**
- **Mitkä tekijät vaikuttavat levyhaun kestoon?**
- **Miten FSCAN poikkeaa SCAN-algoritmista?**
- **Miksi FSCAN?**
- **RAID-levyjen perusideat**
- **Miksi tarvitaan levylohkojen puskurointia?**