LUENTO 14

SIIRRÄNTÄ

Stallings, Luku 11.1-11.4

1

Sisältö

- Yleistä
  - I/O-laitteiden luokittelua, siirtonopeuksia
  - Siirrännän perustekniikat
  - Siirrännän kehittyminen
- DMA-siirto
- Huomioita siirrännästä
  - mm. hierarkia
- Puskurointi
  - Lohkopuskurit

2

Yleistä siirrännästä

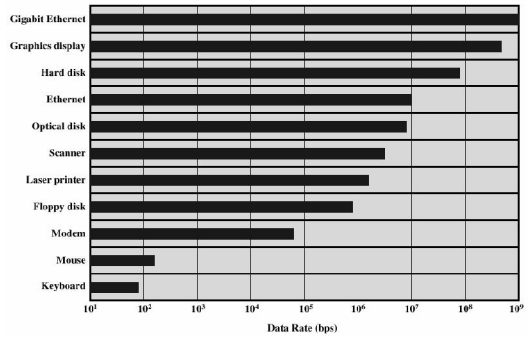
3

I/O-laitteiden luokittelua

- Tieto esitettävä ihmisen / koneen luettavassa muodossa
- Syöttö- ja tulostus
  - näyttö, näppäimistö, hiiri, kirjoitin, ...
  - sensori, skanneri, kamera, mikrofoni ...
- Pysyvä tallennus
  - levy, nauha, CD-ROM ...
- Tiedonsiirto
  - modeemi, verkkokortit, ...
- Eroja nopeudessa, ohjaustarpeessa, siirtoyksiköissä, tiedon esittämisessä ja virhetilanteiden hallinnassa

4

Siirtonopeuksia (teoreettisia) Kuva 11.1



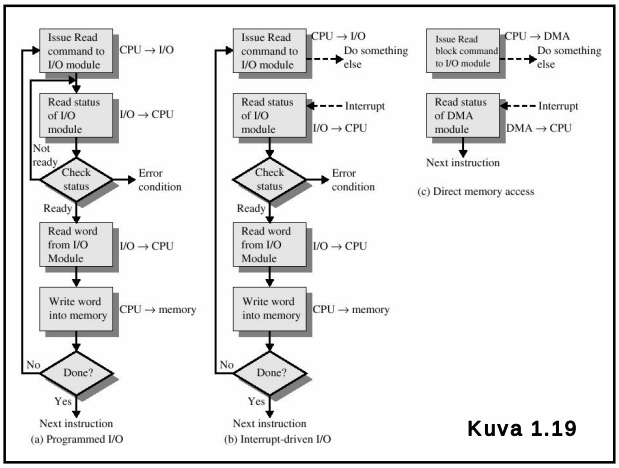
HUOM: Logaritminen asteikko

5

Siirrännän perustekniikat

- Suora I/O (Programmed I/O)
  - CPU tutkii toistuvasti ohjaimen statusrekisteristä onko siirto valmis (busy-wait)
- Epäsuora I/O (Interrupt-driven I/O)
  - CPU antaa siirtotehtävän ohjaimelle ja jatkaa suorittamalla muita prosesseja
  - ohjain keskeyttää, kun siirto valmis
  - CPU siirtää ohjaimen ja muistin välillä
- DMA-siirto (Direct Memory Access)
  - DMA-ohjain osaa siirtää muistin ja laitteen välillä
  - keskeytys vasta, kun koko siirto valmis

6



Kuva 1.19

Siirrännän kehittyminen 1/2

1 CPU (=KJ) huolehtii laitteen ohjauksesta

Lue KJ = laiteajuri

2 Erillinen I/O-ohjain ohjaa yhtä tai useampaa laitetta, suora I/O

Programmed I/O

- CPU (=KJ) tarkkailee koko ajan statusrekisteriä
- Laitteen yksityiskohdat eivät enää KJ:n murheena

3 Ohjain oppii käyttämään keskeytystä

Interrupt-driven I/O

- CPU (=KJ) ei odottele aktivisesti siirron valmistumista

4 Ohjain oppii siirtämään perille saakka = DMA

- CPU (=KJ) vain käynnistää siirron ja tarkistaa onnistumisen

8

Siirrännän kehittyminen 2/2

5 Erillinen I/O-prosessori + DMA

I/O channel

- oma käskykanta
- suorittaa keskusmuistissa olevaa I/O-ohjelmaa
- CPU (=KJ) voi määritellä siirräntätehtävät monipuolisesti
- keskeytys, kun kaikki tehty

6 Erillinen I/O-prosessori + oma muisti + DMA

I/O processor

- ei käytä CPU-väylää I/O-ohjelmansa suorittamiseen
- esim. näytönohjaimella oma prosessori ja muistia

- Yksityiskohdat siirretty KJ:ltä laitteistolle
  - tehokas toteutus

9

DMA
Direct Memory Access

10

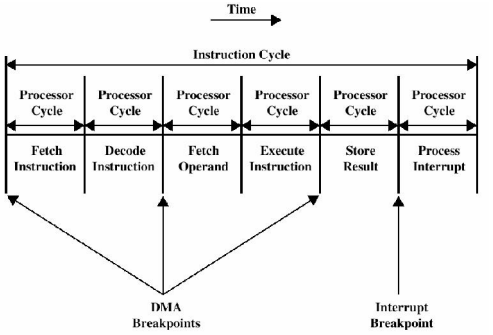
DMA-siirto

- Ohjain osaa siirtää ison määrän tietoa suoraan laitteen ja keskusmuistin välillä
  - Keskeytys vasta, kun koko siirto valmis
- Käyttää muistiin viitatessa prosessoriväylää
- Kilpailu väylän varaamisesta
  - ajoitus s.e. ei häiritse CPU ï ð MEM siirtoja
    - CPU tarvitsee väylää käskynoudossa, operandien noudossa ja tuloksen talletuksessa
  - välimuistin käyttö vähentää väylän käyttöä
- Huom. väylän varaus laitetoiminto, ei aiheuta keskeytystä

11

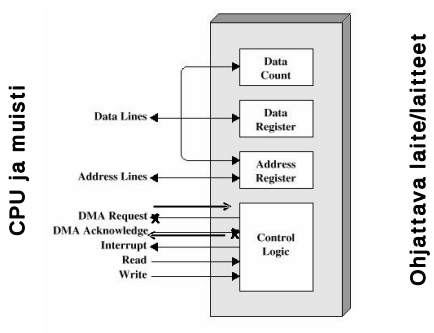
DMA:n ajoitus

Kuva 11.3 [Stal01]



12

DMA-ohjaimen rakenne Kuva 11.2



13

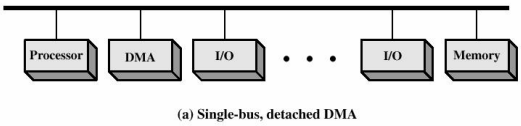
DMA

- Käynnistys: KJ (ajuri) kertoo DMA-ohjaimelle
  - suunta: read vai write
  - mitä I/O-laitetta siirto koskee (jos useita)
    - laiteosoite: esim. levypinta, ura, sektori
  - montako tavua siirretään
  - keskusmuistialueen alkuosoite
- Kun siirto valmis, KJ (ajuri) tarkastaa statuksen
- Prosessoriväylän varaamisia voi vähentää
  - integroimalla DMA-ohjaimen suoraan laiteohjaimeen
  - kytkemällä I/O-laitteet omaan väyläänsä, jolloin niiden välinen siirto ilman CPU-väylän käyttöä

14

Erillinen DMA-ohjain

Fig 11.3 [Stal05]

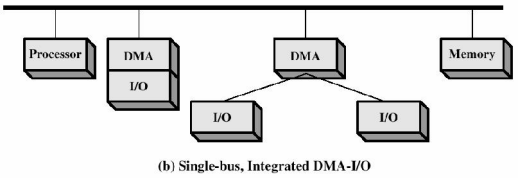


(a) Single-bus, detached DMA

- Kaikki jakavat yhteisen väylän
- DMA-ohjain keskustelee myös laiteohjaimen kanssa käyttäen prosessoriväylää

15

DMA- ja laiteohjain yhdessä



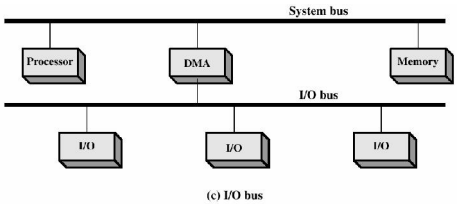
(b) Single-bus, Integrated DMA-I/O

- DMA ei tarvitse prosessoriväylää keskustellessaan laiteohjaimen kanssa
- DMA-ohjain ohjaa yhtä tai useampaa laitetta

Fig 11.3 [Stal05]

16

DMA ja erillinen I/O-väylä



(c) I/O bus

- Suora siirto laitteelta toiselle käyttämättä prosessoriväylää
  - ei häiritse CPU:ta
- Helppo laajentaa

Fig 11.3 [Stal05]

17

Huomioita siirrännästä: hierarkia

18

Huomioita siirrännästä

- Siirräntä järjestelmän suurin pullonkaula
- Siirräntä oheislaitteille erittäin hidasta verrattuna CPU:n ja keskusmuistin välisiin siirtoihin (esim. levy ~ 1 : 1.000.000)
- Siirräntä ei pysy koskaan CPU:n vauhdissa
  - prosessin odoteltava siirräntää
- Moniajon ansiosta CPU voi suorittaa odotusaikana muita prosesseja
- Myös KJ:n tekemä sivutus ja heittovaihto aiheuttaa siirräntää

19

Huomioita siirrännästä

- Levysiirto tärkein tehostettava kohde
  - puskurointi
    - siirrä kerralla enemmän (levylohko)
    - lohkopuskurit (block cache) eli levyvälimuisti (disk cache)
  - ennaltanouto
    - tdsto käsitellään yleensä peräkkäisjärjestyksessä
  - pyyntöjen uudelleenjärjestely
    - minimoi hakuvarren siirrot
    - pyynnöt jonottavat ajurin jonossa

20

Huomioita siirrännästä

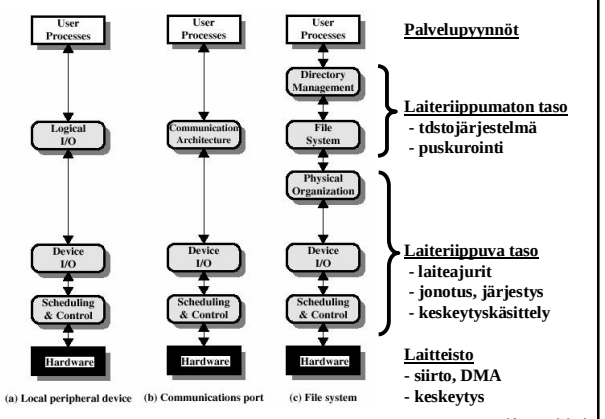
- Siirrännän toteutus yhdenmukaisesti
  - kaikille laitteille samanlainen liitäntä prosessien sekä KJ:n ylimpien tasojen suuntaan
  - myös laitteita käsitellään tdstojärjestelmän kautta (laitetdstot)
- Sovelluksen käyttöön yhdenmukaiset operaatiot kaikille laitteille
  - open(), close(), read(), write(), lock(), unlock() ...
  - kaikki eivät mielekkäitä kaikille laitteille
- Eroavat yksityiskohdat alemmille tasoille
  - loogisen nimen liittäminen fyysiseen laitteeseen
    - tdsto vs. kirjoitin
  - välitasoilla esim. puskurointi ja tiedonsiirron protokollat
  - alimpana varsinaiset laiteajurit
    - ohjaavat laiteohjaimen avulla laitetta

21

Siirrännän hierarkia

- Looginen I/O (laiteriippumaton I/O)
  - prosessi käyttää esim. tdston loogista nimeä
  - tdston rakenne = jono peräkkäisiä tavuja
  - operaatiot: open(), close(), read(), write(), …
- Välitasoilla mm. puskurointi yms.
  - loogisen I/O:n ja fyysisen I/O:n yksiköt erilaisia
    - lue tavu vs. siirrä levylohko
  - tarkista käyttöoikeudet
  - lohkonumero ð levypinta, ura, sektori
  - pyyntöjen järjestely (hakuvarren siirtojen minimointi)
- Fyysinen I/O
  - siirrä hakuvartta, DMA ...

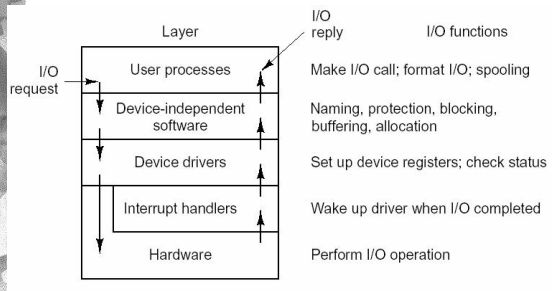
22



Kuva 11.4

Siirrännän hierarkia

Tan01 Kuva 5-16



24

Laiteajurit

- Erityyppisille laitteille omat ajurinsa
- Etsi ajuri laitenumeron perusteella laitekuvaajalistasta
  - siirtoa käynnistettäessä
  - siirron päättyessä (keskeytys!)
- Laitekuvaaja
  - laitteen tunnistus, device id
  - tilatietoa, kenelle laite varattu
  - mitä ajuria käyttää
  - mitä ajurin funktiota (handler) kutsuttava missäkin tilanteessa
    - open(), read(), write(), close() ..., keskeytys
  - jono pyynnöistä parametreineen
    - mm. linkki pyynnön tehneen prosessin PCB:hen

25

Puskurointi

26

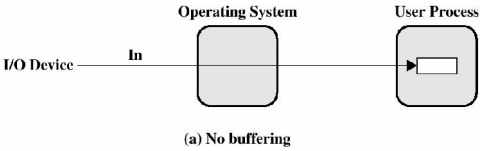
Siirrännän puskurointi

- Tarve
  - Prosessi odottaa Blocked-tilassa siirron valmistumista
  - Alue, jonne siirretään oltava silti muistissa
- Lohkoperustainen
  - Levyt, nauhat
  - kirjanpito vapaasta / varatusta tilasta lohkoittain
  - siirto laitteen ja muistin välillä lohko kerrallaan
  - hajakäsittely mahdollista (nauha?)
- Tavuperustainen
  - pääteyhteys, kirjoitin, hiiri, tiedonsiirtolinja, ...
  - tiedon käsittely tavu kerrallaan
  - vain peräkkäiskäsittely

27

Ei puskurointia

Fig 11.5 [Stal05]



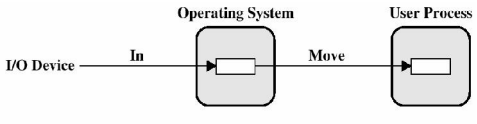
(a) No buffering

- Siirto suoraan prosessin alueelle!
  - Prosessin muistialuetta ei voi heittovaihtaa (lukittava!)
- Tieto käsiteltävä samankokoisina yksikköinä prosessissa ja laitteella
- Käyttöä esim. reaaliaikajärjestelmissä

28

Yksi puskuri

Fig 11.5 [Stal05]



(b) Single buffering

- Ohjain siirtää tiedon KJ:n puskuriin
- KJ siirtää ohjelman alueelle (=muuttujaan)

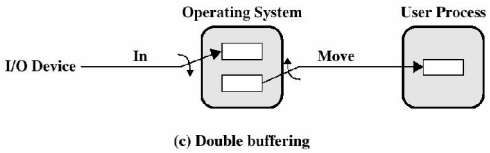
29

Yksi puskuri

- Ennaltanouto (read-ahead)
  - KJ voi täyttää puskurin etukäteen
  - koska peräkkäiskäsittely yleisintä
- Viivästetty kirjoitus (delayed write)
  - prosessin kirjoittama data kootaan puskuriin
  - laitteelle vasta, kun puskuri täysi, tai kun prosessi sulkee laitteen (viim. lohko voi olla vajaa)
- Prosessin voi heittovaihtaa kokonaan
  - siirräntä käyttää aina KJ:n aluetta
- Sopii sekä lohko- että tavuperustaiseen käyttötapaan

30

Kaksoispuskurointi

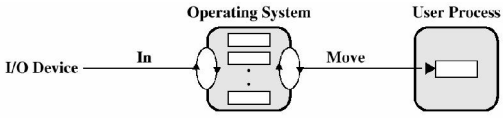


(c) Double buffering

- Kun prosessi käsittelee toisessa puskurissa olevaa tietoa, ohjain lukee toiseen
- Kun ohjain kirjoittaa puskurista laitteelle, prosessi voi täyttää toista puskuria
- Tuottaja - puskuri - kuluttaja -> synkronointi

31

Puskurien käyttö renkaana



(d) Circular buffering

- Jos tuottajan ja kuluttajan nopeudessa satunnaisia eroja, useamman kuin kahden puskurin käytöstä voi olla hyötyä
- esim. verkkoyhteydet

32

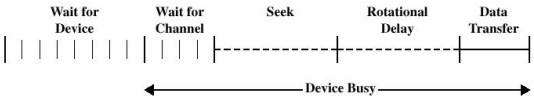
Levypyyntöjen järjestely

Ch 11.5 [Stal 05]

33

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

Näihin voi vaikuttaa

saantiaika (access time)

Tähän ei voi vaikuttaa

?

34

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; hakujonossa 55, 58, 39, 18, 90, 160, 150, 38, 184

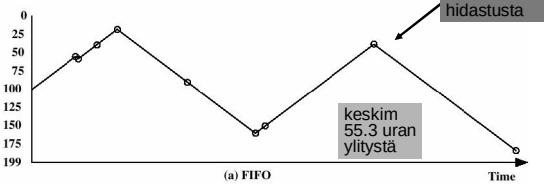
35

First-in-first-out, FIFO

Tbl 11.2 [Stal 05])

- 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

ei mukana hakuvarren kiihdytystä tai hidastusta



keskim 55.3 uran ylitystä

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

(Fig 11.7 [Stal 05])

36

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

37

LIFO, Last-in-first-out

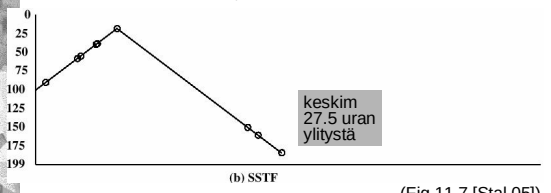
- Palvele viimeksi tullut pyyntö ennen muita
  - anna laite aina aktiivisimmalle prosessille
    - idea: vähentää hakuvarren siirtotarvetta, erityisesti peräkkäistiedostoja käsiteltäessä (paikallisuusilmiö)
- Nälkiintymisvaara
  - jos paljon I/O-sidonnaisia prosesseja, vanha pyyntö voi jäädä jalkoihin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

38

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 ⟶ ...

Tbl 11.2 [Stal 05])



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

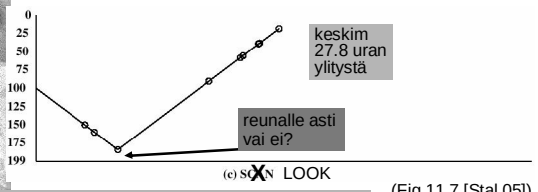
(Fig 11.7 [Stal 05])

39

Tbl 11.2 [Stal 05])

SCAN ja LOOK

- 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



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

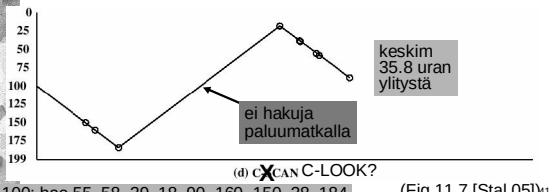
(Fig 11.7 [Stal 05])

40

Tbl 11.2 [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



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

(Fig 11.7 [Stal 05])

41

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

42

N-step-SCAN

- Ongelma: 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 ⟶ FIFO
    - n = ∞ ⟶ SCAN
  - miksi nimi N-step-SCAN?

Levyn vuorotusalgoritmeja

Table 11.2 Comparison of Disk Scheduling Algorithms

(a) FIFO (starting at track 100)		(b) SSTF (starting at track 100)		(c) SCAN (starting at track 100, in the direction of increasing track number)		(d) C-SCAN (starting at track 100, in the direction of increasing track number)	
Next track accessed	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**
55	45	90	10	150	50	150	50
58	3	58	32	160	10	160	10
39	19	55	3	184	24	184	24
18	21	39	16	90	94	18	166
90	72	38	1	58	32	38	20
160	70	18	20	55	3	39	1
150	10	150	132	39	16	55	16
38	112	160	10	38	1	58	3
184	146	184	24	18	20	90	32
Average seek length	55.3	**Average seek length**	27.5	**Average seek length**	27.8	**Average seek length**	35.8

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 04])

- 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 (tässä *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
- Joissakin levyissä voidaan kysellä todellista sektorien sijaintia
- Joissakin levyissä levyohjain toteuttaa omaa optimointiaan
  - ottaa huomioon myös loogiset sektorit
  - ajuri ei silloin optimoi (toivottavasti)