

LUENTO 7

# MUISTINHALLINTA

## Stallings, Luku 7

1

## Sisältöä

- Yleistä muistinhallinnasta (luku 7.1)
- Yksinkertainen muistinhallinta
  - a) kiinteät partitiokoot (luku 7.2)
  - b) dynaamiset partitiokoot (luku 7.2)
  - c) Buddy System (luku 7.2)
  - d) yksinkertainen sivutus (luku 7.3)
  - e) yksinkertainen segmentointi (luku 7.4)

Yksinkertainen =  
prosessi aina kokonaan muistissa tai kokonaan levyllä

2

## YLEISTÄ MUISTINHALLINNASTA

3

## Fyysinen muisti

- Fyysinen muisti, 'sitä voi potkaista'
  - muistiavaruus
  - laitteisto (MMU, väylät) käyttää fyysisiä osoitteita
- Keskusmuisti
  - koodin + käsiteltävän datan suoritusaik. tallennus
  - joukko peräkkäisiä tavuja
- Tukimuisti
  - tiedon (ohjelmat, data) pysyvä tallennus
  - joukko peräkkäisiä lohkoja
- Siirto näiden välillä muistinhallinnan ja tiedostojärjestelmän leipätyötä
  - mahd. automaattisesti KJ:n toimesta

4

## Looginen muisti

- prosessin osoiteavaruus
  - loogiset osoitteet eli virtuaaliosoitteet
- voi olla suurempi kuin fyysinen muisti
- kullakin prosessilla oma osoiteavaruus
  - osoitteet suhteellisia alun suhteen (0..MAX)
- sovellus jakaantuu loogisesti moduuleihin
  - ne voidaan tehdä eri aikoina, osoitteiden paikkaus linkityksessä
  - erilaisia käyttöoikeuksia
    - koodi R (vapaakäytisyys)
    - data R / W / RW
  - osa moduuleista tarkoitettu yhteiskäyttöön

5

## Prosessin rakenne

- PCB
- Ohjelma-alue
- Data-alue
- Pino alueen lopussa
- Suhteellisia viittauksia prosessin omalla muistialueella

The diagram illustrates the memory layout of a process. It shows a vertical stack of components: Process Control Block (PCB) at the top, followed by Program, Data, and Stack. Arrows indicate the following relationships:

- 'Process control information' points to the PCB.
- 'Entry point to program' points to the start of the Program section.
- 'Increasing address values' points downwards, indicating that addresses increase from the Program section towards the Stack.
- 'Current top of stack' points to the top of the Stack section.
- 'Branch instruction' points to a location within the Program section.
- 'Reference to data' points to a location within the Data section.

Figure 7.1 Addressing Requirements for a Process

## Yleistä muistinhallinnasta

- Ytimelle voidaan sallia kiinteä paikka muistissa
  - voi käyttää suoraan fyysisiä osoitteita
- Loppu KJ:n muille osille ja sovelluksille
- KJ huolehtii siitä, että muistiin mahtuu mahdollista prosessia
  - vapaan / varatun tilan hallinta
- Laitteisto huolehtii siitä, etteivät prosessit sotke toisiaan
  - MMU

7

## Muistinhallinta: vaatimuksia

### Vapaa sijoitettavuus

- Heittovaihto
  - KJ voi siirtää prosesseja välillä levyille
  - KJ voi ottaa suoritettavaksi useampia prosesseja
  - prosessin paikka voi vaihdella suorituksen aikana
- Ohjelmoija ei voi tietää minnepäin muistia sovellus sijoittuu suoritusajana
  - suhteelliset osoitteet
- Viittaukset fyysisiksi osoitteiksi viimeistään ennen muistinoutoa/talletusta

⇒ Ajonaikainen osoitemuunnos MMU:ssa

8

## Muistinhallinta: vaatimuksia

### Suojaus

- Luvatta ei saa käyttää toisen muistialueita
- Osoitetarkistus käännoaikana mahdotonta
  - sovelluksen moduulit voidaan kääntää eri aikoina
  - prosessien sijainti voi vaihtua suoritusajana
  - käskykanta voi sallia osoitustapoja, joissa osoite lasketaan suoritusajana
- KJ ei voi sitä tehdä!

⇒ Ajonaikainen laillisuustarkistus osittain MMU:ssa ja osittain KJ:ssa

9

## Muistinhallinta: vaatimuksia

### Yhteiskäyttö

- Sallittava yhteisen koodin / datan käyttö
  - suojauksista tinkimättä!
- Koodi ei muutu suorituksen kuluessa
  - vapaakäyntisyys (reentrancy)
  - järkevämpää sallia koodin yhteiskäyttö kuin pitää muistissa useita kopioita
- Monet prosessit tekevät yhteistyötä muiden kanssa, joten niillä yhteisiä tietorakenteita
  - Esim. tuottajalla ja kuluttaja yhteinen puskuri

⇒ käytä säikeitä, palvelupyynnöitä

10

## YKSINKERTAINEN MUISTINHALLINTA

11

## Yksink. muistinhallinta

- = Prosessi kokonaan muistiin / levyille
- Jos ohjelma > fyysisen muistin koko, sitä ei välttämättä pystytä suorittamaan
  - Menetelmät:
    - a) kiinteä partitiointi
    - b) dynaaminen partitiointi
    - c) Buddy System
    - d) yksinkertainen segmentointi
    - e) yksinkertainen sivutus
  - Katso ns. katoava kansanperinne
    - ei paljon käyttöä nykyisissä järjestelmissä
    - kuuluu kuitenkin KJ:n peruskäsitteistöön

12

## a) Kiinteät partitiot

13

## Kiinteät partitiot

- KJ (operaattori) jakoi muistin kiinteänkokoisin partitioihin
- Varausalueet olivat kaikki yhtäsuuria tai niiden koot saattoivat vaihdella
- Ohjelma, joka oli pienempi tai yhtäsuuri kuin partitio, voitiin ladata ja ajaa ko. partitiossa

Equal-size partitions      Unequal-size partitions

## Kiinteät partitiot

- Jos ei riittävän suurta vapaata partitiota, KJ teki tilaa heittovaihdolla
  - joku prosessi levyllä
  - PCB jäi muistiin
- Jos ohjelma niin iso, ettei sopinut mihinkään partitioon, piti ohjelmoijan ratkaista tilanne
  - kerrostus (overlying)
    - "ohjelmoijan hoitama segmentointi"
  - vain osa ohjelmasta muistissa
  - piti koodata mitä osia (aliohjelmaa, 'segmenttejä') kullakin hetkellä muistissa ja missä kohdassa

15

## Kiinteät partitiot

- Muistin käyttö melko tehontonta
  - varattiin aina kokonainen partitio, vaikka vähempikin olisi riittänyt
- Sisäinen pirstoutuminen (internal fragmentation)
  - partitioiden sisälle jäi tyhjää tilaa
  - vapaa tila yhdessä olisi saattanut riittää uudelle prosessille, mutta se ei ollut yhtenäisellä alueella

16

## Kiinteät partitiot: Sijoitus

- Jos kaikki partitiot samankokoisia
  - jos vapaita, valitse joku niistä
  - jos kaikki varattuja, heittovaihdolla joku Blocked-prosessi levyllä
    - tilaa vapautuu aina saman verran
- Jos partitiot eri kokoisia
  - valitse pienin partitio, johon prosessi sopii
    - yrityksi minimoida sisäistä pirstoutumista
  - a) erikokoisille partitioille omat prosessijononsa
    - partitiokokoon yksi työn parametreista
  - b) yksi jono, josta valittiin johonkin vapaaseen partitioon

17

## Kiinteät partitiot: Sijoitus

Kuva 7.3

18

## b) Dynaaminen partitiointi

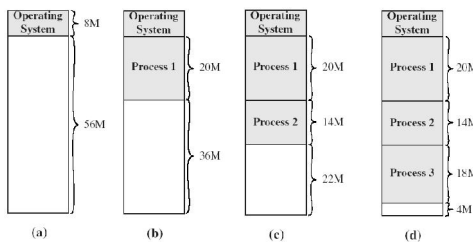
19

## Dynaaminen partitiointi

- Ei etukäteen partitiointiä
- Varausten koot ja lukumäärä vaihtelivat dynaamisesti prosessien tarpeiden mukaan
- Prosessille muistia vain sen verran kuin tarvitsi
- Ulkoinen pirstoutuminen (**external fragmentation**)
  - varausten / vapautusten tuloksena väleihin jäi pieniä vapaita alueita
- KJ tiivistä muistia välillä (**compaction**)
  - prosesseja siirrettiin, jotta vapaa tila yhteen kohtaan
  - yleisrasite

20

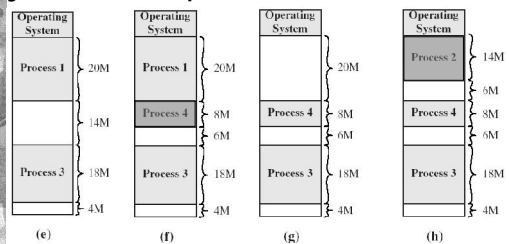
## Dynaaminen partitiointi Kuva 7.4



- Vaiheessa (d) ei enää tilaa prosessille 4, koko 8 M
- Jos kaikki 3 prosessia joutuivat Blocked-tilaan, KJ otti uuden P4:n suoritettavaksi

21

## Dynaaminen partitiointi



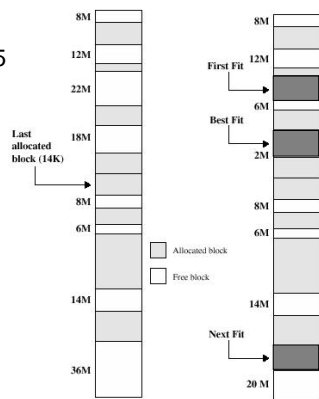
- Niinpä KJ heitti P2:n ulos (e) ja P4 sai sen paikan (f)
- Aikanaan P2 pääsi taas Ready-tilaan
- Kun muistissa ei Ready-prosesseja, KJ vaihtoi P2:n muistiin
- Tuloksena 3 pirstalletta:  $6M + 6M + 4M = 14M$

22

## Sijoitus Kuva 7.5

Mistä kohtaa varataan?

- Tavoitteena vähäinen tiivistämistarve
- Best-fit kooltaan sopivin
- First-fit ens. kooltaan riittävän suuri
- Next-fit jatka etsintää edellisestä kohdasta



Example Memory Configuration Before and After Allocation of 16 Mbyte Block

## Paras sijoitusalgoritmi?

- First fit
  - yksinkertainen
  - helpoin toteuttaa tehokkaasti (paras)
  - isosta aukosta jää jäljelle isohko aukko
- Best fit
  - hyvä nimi, mutta tuloksena mahdoll. pieniä aukkoja
  - tuloksena nopeimmin pirstoutuminen
- Next fit
  - varaukset / aukot muistin loppuosaan
- Toteutus: linkitetty lista, jossa vapaat alueet koon tai osoitteen mukaisessa järjestyksessä
  - kumpi parempi?
  - (osoite,pituus)ä ... ä (osoite,pituus)ä

24

## Poistoalgoritmi

- Kun muistissa vain Blocked-prosesseja, kannatti ottaa uusi prosessi suoritukseen
  - ettei CPU jouten
  - heittovaihdta joku levyille
  - poistettava prosessi Blocked-Suspend-tilaan
  - uusi prosessi tilaan Ready-Suspend tai Ready
- Mikä pois, jotta saatiin sopivasti vapaata?
  - iso vs. pieni
  - sellainen, jonka vieressä iso tyhjä alue
  - pieni vs. suuri prioriteetti

25

## c) Buddy System

26

## Buddy System

- Kompromissi kahdesta em. menetelmästä
  - kiinteät partitiokoot, mutta dynaaminen jako partitioihin
  - ei haluta jättää pieniä tyhjiä tiloja
  - varaa pikkuisen enemmänkin kuin tarve vaatii
- Varausyksikön koko ilmaistaavissa 2:n potenssissa
  - suurin mahdollinen yleensä koko muisti
  - pienimmälle varausyksikölle joku minimikoko
    - yksinkertaistaa vapaiden alueiden kirjanpitoa
- Varaukset ja vapaiden alueiden yhdistely toteutettavissa tehokkaasti

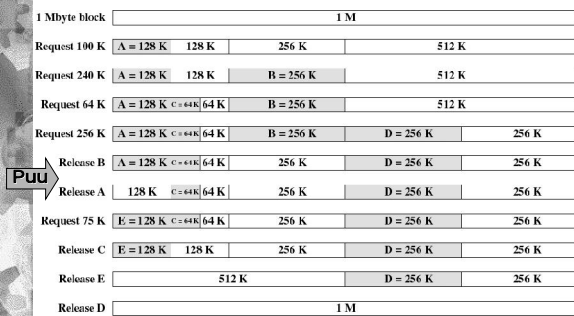
27

## Buddy System

- KJ ylläpitää kustakin koosta omaa listaa
  - lista osoitteiden mukaisessa järjestyksessä
  - aluksi vain yksi suuri varausyksikkö
- Varaus
  - jos oikean kokoluokan listassa ei vapaata alkiota, jaa luokkaa suurempi alue kahdeksi pienemmäksi
  - toista tarvittaessa
- Vapautus
  - kun listassa kaksi fyysisesti vierekkäistä aluetta, yhdistä ne kokoluokkaa suuremmaksi alueeksi
  - toista tarvittaessa

28

## Buddy System: esimerkki



Kuva 7.6

29

## Buddy: varaukset puuna

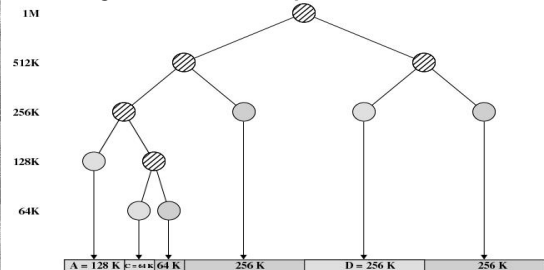


Figure 7.7 Tree Representation of Buddy System

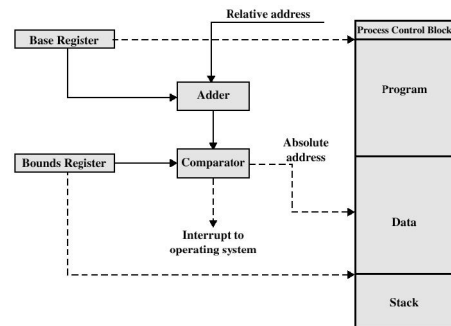
30

### a) b) c) Osoitemuunnos

- Ohjelmassa loogiset osoitteet
  - kaikki suhteellisia sen alun suhteen
- Historia  
Lataaja (KJ:n osa) muutti loogiset osoitteet fyysisiksi osoitteiksi samalla, kun latsi ohjelmakoodin muistiin
- Nykyaika  
osoitemuunnos vasta käskyjä suoritettaessa
- KJ:n ydin voi käyttää fyysisiä osoitteita, joten kun CPU suorittaa KJ:tä, osoitemuunnosta ei tarvitse tehdä

31

### a) b) c) Osoitemuunnos Kuva 7.8



32

### Laitteistotuki

- MMU:hun muunnosta ja suojaustarkistusta varten kaksi rekisteriä
  - ns. kanta- ja rajarekisteri
  - Base prosessin fyysinen alkuosoite
  - Limit prosessin loppuosoite (tai pituus)
- Kun prosessi suoritukseen, kopioidaan näille arvot PCB:stä
- $Fyys.osoite = loog.osoite + Base$   
jos fyysinen osoite > Limit  
aiheuta poikkeus 'virh. muistiosoite'
- muuten  
MAR  $\leftarrow$  fyysinen osoite

33

### d) Yksinkertainen sivutus

34

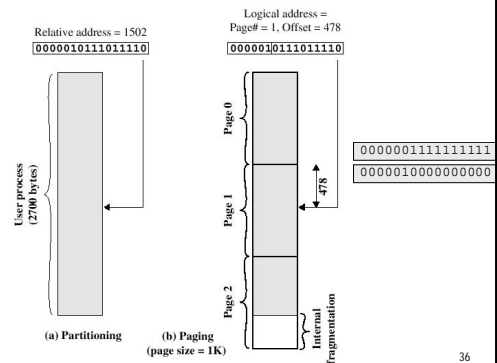
### Yksinkertainen sivutus

- KJ varaa muistia sivutila kerrallaan (**page frame**)
  - kaikki samankokoisia
  - 'suhteellisen pieniä', esim. 1KiB tai 4 KiB
  - koko aina joku 2:sen potenssi
    - käsittely helppoa laitetasolla
- KJ käsittelee ohjelmaa sivuina (**page**)
  - sivu ja sivutila samankokoisia
- KJ sijoittaa prosessin sivut vapaisiin sivutiloihin
  - Kun käytössä yksinkertainen muistinhallinta, KJ tuo kaikki sivut kerralla muistiin / vapauttaa

35

### Sivutus

Kuva 7.11



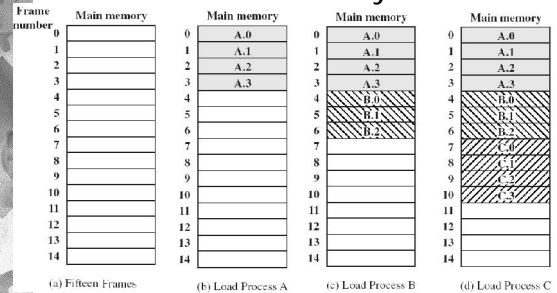
36

## Yksinkertainen sivutus

- KJ pitää prosessikohtaista sivutaulua (**page table**)
  - PCB:ssä sivutaulun fyysinen alkuosoite
    - osoite MMU:hun, kun prosessi suoritukseen
  - alkiossa tieto sivutilasta, jossa ko. sivu majoilee
- Looginen osoite muodostuu nyt parista (sivunumero, siirtymä)
  - bitinjonon alkupään bitit kertoo sivunumeron
  - bitinjonon loppupään bitit siirtymän sivun sisällä
- Vain viimeinen sivu voi aihe. pirstoutumista

37

## Esimerkki muistin käytöstä

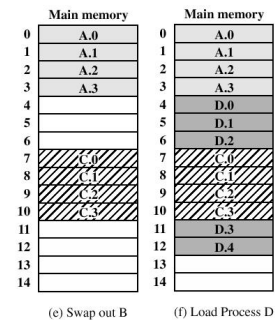


- (d) Prosessi D (5 sivua) ei sovi muistiin

38

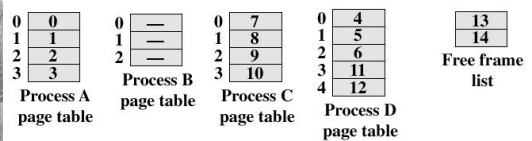
## Esimerkki (jatkuu) Kuva 7.9

- (e) KJ heittävähtää prosessin B levyllä
- (f) KJ lataa prosessin D muistiin (5 sivua)
- Prosessin D sivut eivät sijaitse peräkkäin
- Ei ulk. pirstoutumista
- Sis. pirstoutumista keskim. 1/2 sivutilaa per prosessi



39

## Esimerkin prosessien sivutaulut

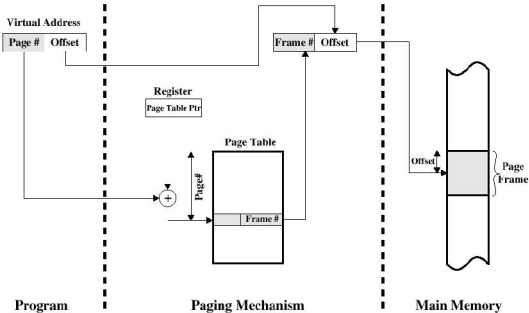


- Jokaisella prosessilla ikeyma sivutaulu
  - Sivutaulun alkiossa sen sivutilan numero, jossa sivu sijaitsee
- KJ:llä sivutilataulu (tai linkitetty lista), josta käy ilmi mitkä sivutilat vapaita

40

## Osoitemuunnos (siv.)

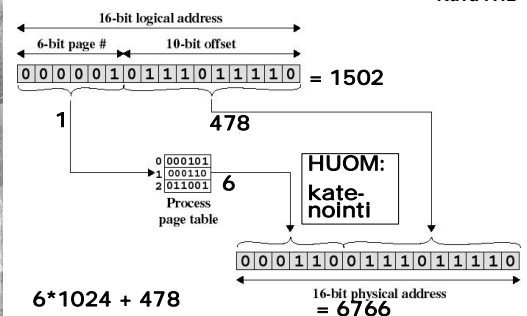
Kuva 8.3



41

## Looginen vs Fyysinen osoite

Kuva 7.12



$$6 * 1024 + 478 = 6766$$

42

## e) Yksinkertainen segmentointi

43

## Yksinkertainen segmentointi

- Ohjelmoija tai kääntäjä jakaa ohjelman kooltaan erilaisiin loogisiin kokonaisuuksiin, segmentteihin
  - segmentti = esim. data-alue tai muutama aliohjelma
- Kääntäjä tuottaa koodia, jossa segmentin sisäiset loogiset osoitteet
  - osoite tavallaan muotoa (segmentti, siirtymä)
    - alkupään bitit kertoo segmenttinumeron
    - loppupään bitit kertoo siirtymän segm. sisällä
- Järjestelmässä yleensä segmentin maksimikoko

44

## Segmentointi

Kuva 7.11

45

## Yksinkertainen segmentointi

- Kun KJ lataa prosessin muistiin, se voi sijoitella segmentit vapaasti muistiin
  - kun käytössä yksinkertainen muistinhallinta, KJ tuo kaikki segmentit kerralla muistiin
  - muistia varataan aiemmin esitetyillä menetelmillä segmentikohtaisesti
- KJ ylläpitää prosessin segmenttitaulua
  - PCB:ssä segmenttitaulun fyysinen alkuosoite
    - osoite MMU:hun, kun prosessi suoritukseen
  - alkioissa kunkin segmentin fyysinen alkuosoite (Base) ja pituus (Length)

46

## Osoitemuunnos (segm.)

Kuva 8.12

47

## Looginen vs Fyysinen osoite

48

### Arviointia

- Segmenttitaulun alkiossa alkuosoite ja pituus
  - segmentin kokoa helppo kasvattaa/pienentää dynaamisesti
    - saattaa vaatia segmentin uudelleensijoittamista
  - osoitteen oikeellisuus tarkistettavissa MMU:ssa
- Segmentti sopiva suojauksen yksikkö
  - ohjelmoija määrittelee segmentit ja käyttöoikeudet
  - käyttötapa kopioitu segmenttitaulun alkioon
- Ongelmana muistin pirstoutuminen
  - segmentit eri kokoisia
  - muistin tiivistämistarve

49

### Segmentointi vai Sivutus

- Pirstoutuminen
  - segmentointi aiheuttaa ulkoista, sivutus sisäistä
  - segmentoinnin yhteydessä muistin tiivistämistä
- Osoitemuunnos MMU:ssa
  - rekisteri, jossa segmentti-/sivutaulun fyys. osoite
  - segmentoinnissa yhteenlaskua (yht.laskulaitteistoa)
  - sivutuksessa katenoitua (helppoa)

50

### Segmentointi vai Sivutus

- Sivutus ohjelmoijalle näkymätöntä, segmentointi ei
- Segmentti suojauksen kannalta loogisempi yksikkö kuin sivu
- Segmentointi sopii hyvin rutiinien dynaamiseen linkitykseen (yhteiskäyttö helpompi järjestää)
- Miksei molemmat: Yhdistettävissä!

51

### Yhteenveto

Menetelmä	kuvaus	vahvuudet	heikkoudet
Kiinteä partitio	Muisti jaettu etukäteen osiin. Prosessi vain yhdessä osassa.	helppo toteutus	sisäinen pirstoutuminen maksimi prosessimäärä rajoitettu
Dynaaminen partitio	Muistia varataan tarpeen mukaan. Prosessi vain yhdessä osassa.	ei sis. pirst. par. muistin käyttöaste	ulkoinen pirstoutuminen, tiivistämistarve
Buddy System	Muistinvar. dyn., mutta kiinteänkokoisina osina. Prosessi vain yhdessä osassa.	ei juurikaan ulkoista pirstoutumista	vähäinen sisäinen pirstoutuminen
Yks. segmentointi	Prosessi jaettu segmentteihin. Segm. sijoitettavissa vapaasti.	ei sis. pirst. par. muistin käyttöaste	ulkoinen pirstoutuminen
Yks. sivutus	Prosessi ja muisti jaettu sivuihin. Sij. vapaasti	ei ulk. pirst.	hyvin vähän sis. pirst. (vain viimeinen sivu)

### Binääriset etuliitteet

Factor	Name	Symbol	Origin	SI derivation
2 <sup>10</sup>	kibi	Ki	kilobinary (2 <sup>10</sup> ) <sup>1</sup>	kilo (10 <sup>3</sup> ) <sup>1</sup>
2 <sup>20</sup>	mebi	Mi	megabinary (2 <sup>10</sup> ) <sup>2</sup>	mega (10 <sup>3</sup> ) <sup>2</sup>
2 <sup>30</sup>	gibi	Gi	gigabinary (2 <sup>10</sup> ) <sup>3</sup>	giga (10 <sup>3</sup> ) <sup>3</sup>
2 <sup>40</sup>	tebi	Ti	terabinary (2 <sup>10</sup> ) <sup>4</sup>	tera (10 <sup>3</sup> ) <sup>4</sup>
2 <sup>50</sup>	pebi	Pi	petabinary (2 <sup>10</sup> ) <sup>5</sup>	peta (10 <sup>3</sup> ) <sup>5</sup>
2 <sup>60</sup>	exbi	Ei	exabinary (2 <sup>10</sup> ) <sup>6</sup>	exa (10 <sup>3</sup> ) <sup>6</sup>
2 <sup>70</sup>	zebi	Zi	zettabinary (2 <sup>10</sup> ) <sup>7</sup>	zetta (10 <sup>3</sup> ) <sup>7</sup>
2 <sup>80</sup>	yobi	Yi	yottabinary (2 <sup>10</sup> ) <sup>8</sup>	yotta (10 <sup>3</sup> ) <sup>8</sup>

- Tarve: Koon mukana kasvava virhe vastaaviin desim. esityksiin: k ~ 2,5%, M ~ 5%, G ~ 7,5%

53