

# SÄIKEET, SMP

Stallings, Luku 4

1

## Sisältöä

- Prosessi vs. säie
- Miksi säikeitä?
- ULT: Käyttäjätason säikeet
- KLT: Säikeiden toteutus ytimessä
  
- SMP
- Windowsin säikeet

2

## Säikeet (multithreading)

- Prosessi voi jakautua yhteen tai useampaan säikeeseen, jotka yhtäaikaan Ready-jonossa
- *Yhden prosessin säikeet käyttävät yhteistä koodia, data-aluetta ja resursseja sekä pääosaa prosessin kuvaajasta*
- *Säikeen luonti ja lopettaminen nopeampaa kuin prosessin*
- *Saman prosessin säikeiden vuorottaminen nopeampaa kuin eri prosessien vuorottaminen*

3

PROSESSI vs SÄIE

4

## Prosessi perinteisesti

- Resurssien omistaja, jolle allokoitu
  - virtuaaliosoiteavaruus, eli suoritussympäristö
    - prosessin kuva (image): PCB, koodi, data, pino
  - resursseja
    - muistia, tiedostoja, I/O-laitteita ...
- Vuorottajan hallinnoima kokonaisuus - prosessi on ohjelman suoritus koneessa
  - suoritus limittäin muiden prosessien kanssa
  - prosessiin liittyy tila (Running, ...) sekä prioriteetti

**Process, task**

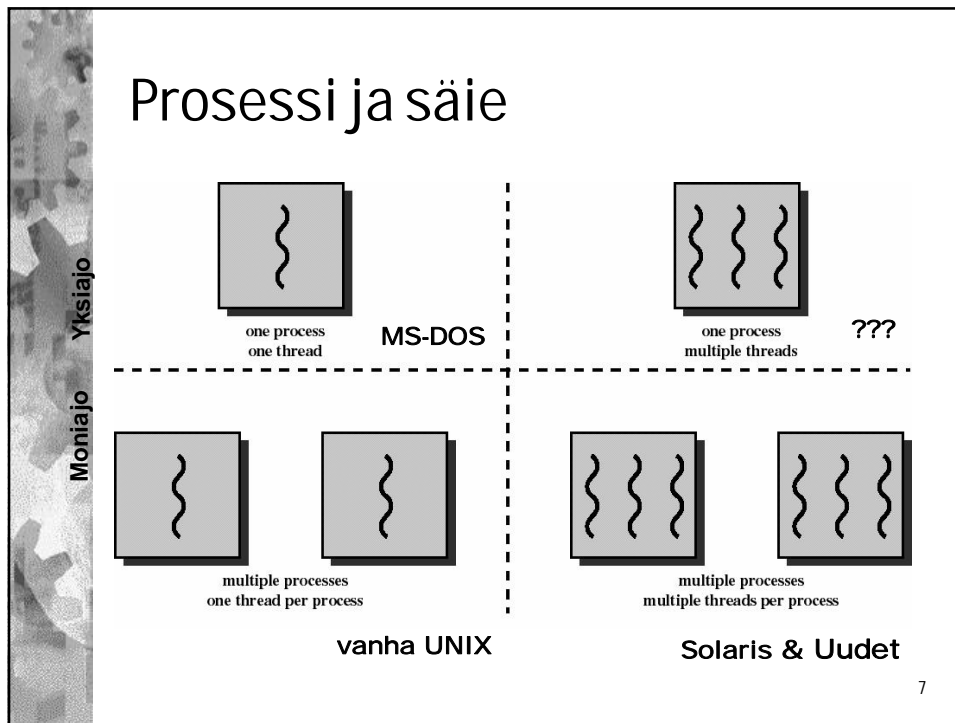
5

## Prosessi nykyaikaisesti

- Resurssien kirjanpidon yksikkö, omistaja
    - virtuaaliosoiteavaruus, jossa prosessin kuva
    - laitteiden varaus
  - Suojauksen yksikkö
    - muistinsuojaus
    - prosessien välinen kommunikointi
    - tdstot ja niiden pääsyoikeudet
- mutta Vuorottamisen yksikkö = Säie
- CPU suorittaa säikeitä, ei prosesseja
  - Yksi koodi + resurssit, monta suoritusta

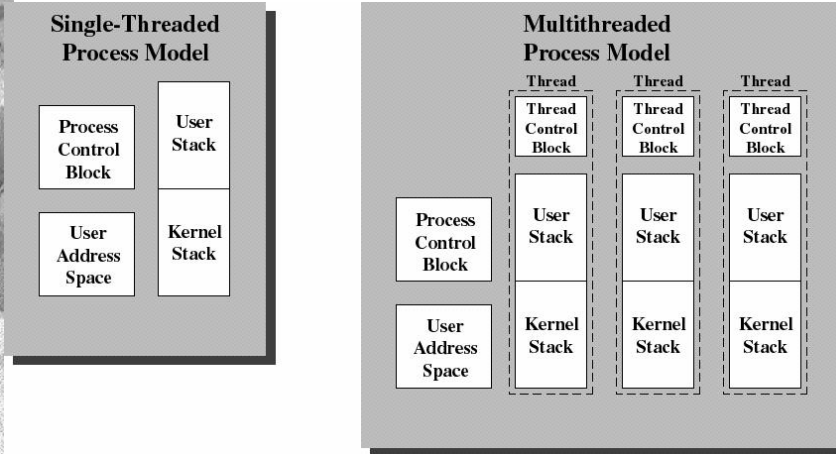
**Thread, Lightweight process**

6



- ## Säie
- Säikeellä oma *tila* (Running, Ready...)
  - Säikeellä oma *tallealue rekistereille*
    - mm. omat PC:n ja PSW:n arvot
  - Säikeellä oma *pino*
    - aliohjelmakutsuja ja paikallisia muuttujia varten
  - è Säikeellä oma *kuvaaja*
    - TCB, Thread Control Block
- 8

## Yksi säie vs. Monta säiettä



**Säikeen kuvaaja TCB**  
tallealue rekistereille, prioriteetti, tila, ...

9

## Säikeen tilat

- Perustilat: Running, Ready, Blocked
  - kuten prosessilla
- Suspend-tila koskee aina koko prosessia
  - liittyy heittovaihtoon
  - osoiteavaruus prosessitason käsite
  - mm. koodialue + globaali data yhteiskäytössä
- Kun prosessi joutuu Suspend-tilaan, joutuvat kaikki sen säikeet odottamaan
- Kun prosessi lopetetaan, poistetaan samalla kaikki sen säikeet

10

## Yhteiskäyttöiset resurssit

- Prosessin säikeet käyttävät yhteistä koodi- ja data- aluetta
  - viite sivutauluun vain PCB:ssä
  - mutta jokaisella oma suoritusaikainen pino
- Kun säie muuttaa data-aluetta (muuttujia), muutos näkyy kaikille prosessin säikeille
- Säikeen avaama tdsto avoinna myös muille prosessin säikeille
  - tiedostokuvaajataulu vain PCB:ssä
  - yhteinen luku/kirjoituspositio?

11

## Miksi säikeitä?

- ⊕ Säikeen luonti (> 10 x) nopeampaa kuin kokonaan uuden prosessin luonti
- Prosessin säikeiden vuorottaminen nopeampaa kuin prosessien vuorottaminen
- ⊕ Kun säie odottaa, voidaan suorittaa jotain muuta saman prosessin säiettä
- Säikeen lopettaminen nopeampaa kuin prosessin lopettaminen
- Voi helpottaa ohjelmointityötä

12

## Miksi säikeitä?

- ' Resurssien jakaminen säikeiden välillä tehokasta
  - oletus: yhteiskäyttöisiä
- ' Saman prosessin säikeiden välinen kommunikointi helppoa ja nopeaa
  - yhteiskäyttöinen data-alue
    - kaikilla pääsy globaaleihin muuttujiin
  - säie hoitaa itse tiedon sovittuun paikkaan, toinen noutaa itse
  - ei tarvita ytimen apua, ei siirtymisiä etuoik. tilaan

13

## Miksi säikeitä?

### Mutta

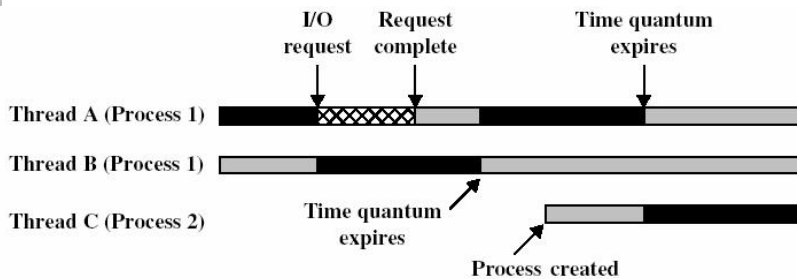
- synkronointi ja poissulkeminen kokonaan ohjelmoijan vastuulla
  - esim. Yhteisen tietorakenteen muuttaminen
    - jos yksi muuttamassa, muut eivät saa käyttää
    - jos väh. yksi käyttää, kukaan ei saa muuttaa
  - esim. Tuottaja ja kuluttaja
    - kuluttaja ei saa edetä ennenkuin tuottaja edennyt tiettyyn vaiheeseen

Ö RIO-kurssi

14

## Kuka hyötyy?

Kuva 4.4



Sovellus, jossa selkeästi riippumattomia kokonaisuuksia

- ts. suoritusjärjestyksellä ei väliä
- ts. säikeillä ei synkronointitarvetta
- kukin osa toteutettavissa omana säikeenä

15

## Kuka hyötyy?

Esim: lähiverkon tdstopalvelija

- Käsiteltävä useita pyyntöjä lyhyessä ajassa
- Halvempaa luoda/tappaa kutakin pyyntöä kohden oma säie kuin oma prosessi
- SMP: prosessin säikeet voidaan suorittaa aidosti yhtäaikaan eri prosessoreilla

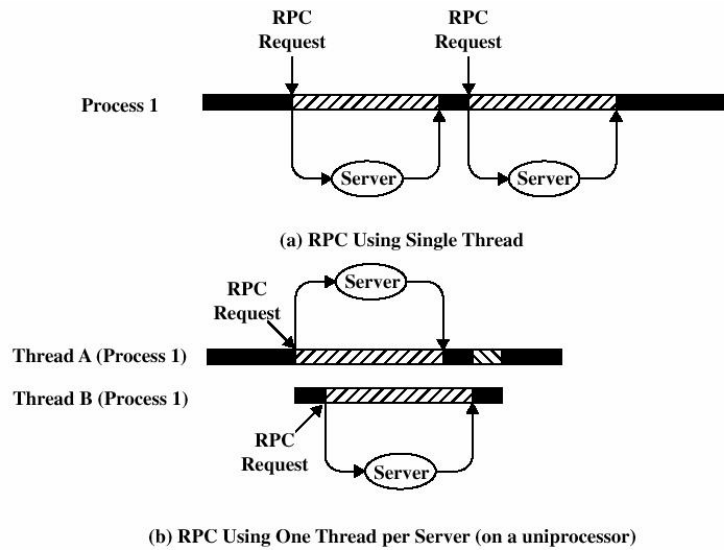
Esim: komennot valikoista

- Yksi säie näyttää valikon ja lukee syötteen
- Toinen säie suorittaa edellistä komentoa
- Helpompi ohjelmoida

16



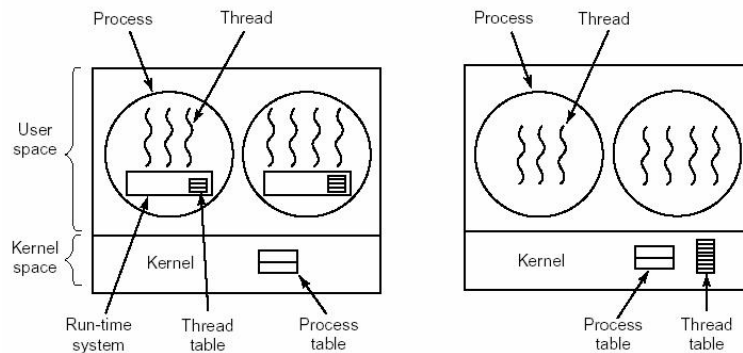
## Esimerkki: etäkutsu Kuva 4.3



17

## Prosessi voi jakautua säikeisiin

- ... jos KJ:ssä toteutettu säikeet tai = KLT
- ... jos käytetään säiekirjastoa = ULT



Tan01 kuva 2-13

18

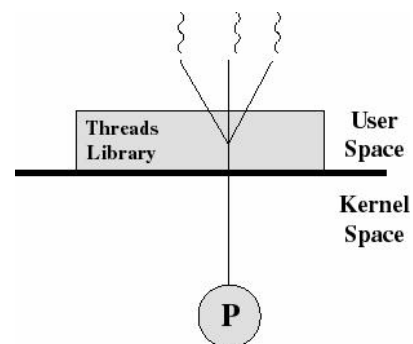
## KÄYTTÄJÄTASON SÄIKEET

ULT, User Level Threads

19

## Käyttäjätason säikeet

- Ydin ei tiedä säikeistä
  - vuorottaa vain prosesseja
- Sovellus käyttää säikeiden hallintaan säiekirjastoa
  - kirjanpito, TCB:t
- Sovellus huolehtii itse säikeidensä vuorottamisen
  - ohjelmoijan vastuulla
  - ei käytä ytimen koodia
  - ei keskeytysohjattua



20

## Säiekirjaston perusrutiinit

### Karkea jako:

- Säikeen luominen
    - kirjanpito säikeestä ja sen tilasta
  - Säikeen etenemisen estäminen
    - rekistereiden tallettaminen
  - Säikeen etenemisen salliminen
    - rekistereiden palauttaminen
  - Säikeen lopettaminen
- } synkronointi

21

## POSIX threads (pthreads)

- pthread-kirjastossa yli 60 funktiota
- pthread\_create()
  - parametrina funktio, josta suoritus alkaa
- pthread\_exit()
  - lopeta säikeen suoritus
- pthread\_join()
  - odota parametrina annetun säikeen loppumista
- Synkronointi, poissulkeminen (semaforit)
  - pthread\_mutex\_init() / \_destroy()
  - pthread\_mutex\_lock() / \_trylock() / \_unlock()
- Ynnä muita funktioita
  - sched\_yield(): luovu vapaaehtoisesti CPU:sta

22

## Esimerkki

```

void main() {
    pthread_t thr1, thr2;
    char *msg1 = "Hello";
    char *msg2 = "World";
    pthread_create(&thr1, pthread_attr_default,
        (void*)&print_message_function, (void*)msg1);

    pthread_create(&thr2, pthread_attr_default,
        (void*)&print_message_function, (void*)msg2);
    exit(0);
}

void print_message_function(void *ptr){
    char *message;
    message = (char *) ptr;
    printf("%s ", message);
}
    
```

23

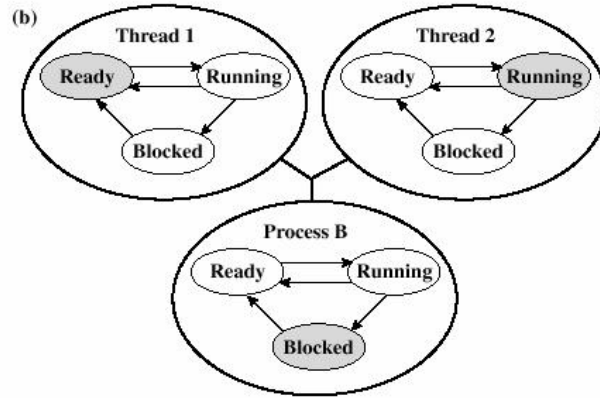
## ULT-säikeen vs Prosessin tila

- KJ:n ydin ei tiedä ULT-säikeistä
  - KJ vuorottaa prosessitasolla
    - Ready-jonossa prosesseja
  - aikaviipale prosessille
  - säiekirjasto huolehtii säikeiden vuorottamisesta prosessin aikaviipaleen sisällä
- Jos ULT-säie odottaa palvelupyynnössä, prosessi joutuu Blocked-tilaan
- mutta säiekirjaston kirjanpidossa säie edelleen Running-tilassa

Ö ULT- säikeen ja prosessin tila erillisiä

24

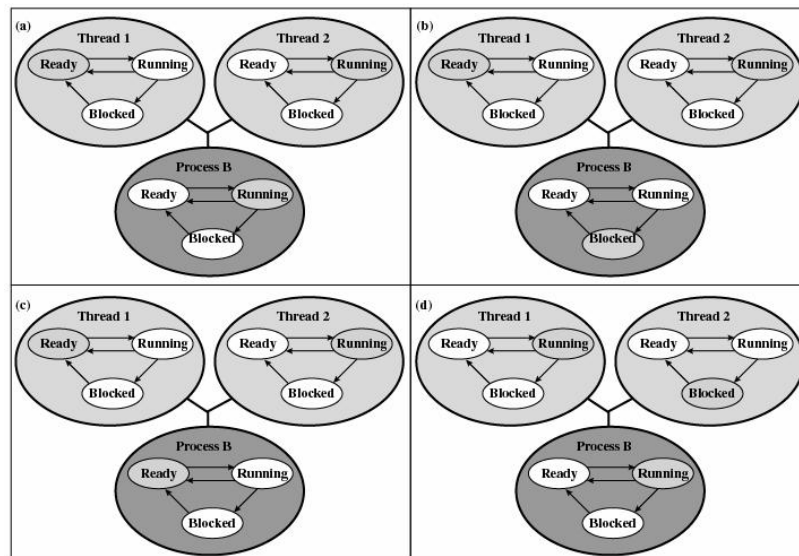
## ULT-säikeen vs Prosessin tila



Kuva 4.7

25

## Mahdollisia tilan vaihtoja



Colored state  
is current state

Figure 4.7 Examples of the Relationships Between User-Level Thread States and Process States

## Käyttäjätason säikeet

### Hyötyjä

- Vuorottaminen nopeaa
  - ei KJ:n apua
  - ei keskeytystä
  - ei prosessinvaihtoa!
- Ohjelmoija voi valita sopivan vuorottamistavan
  - KJ ei tunne sovelluksen tarpeita
- Sovellus siirrettävissä helposti ympäristöihin, joissa sama säiekirjasto

### Haittoja

- Kun säikeen palvelupyyntö aiheuttaa odotusta, kaikki muut saman prosessin säikeet odottavat
- Saman prosessin säikeet eivät voi olla suorituksessa usealla prosessorilla yhtäaikaan
  - Ydin vuorottaa vain prosesseja!

27

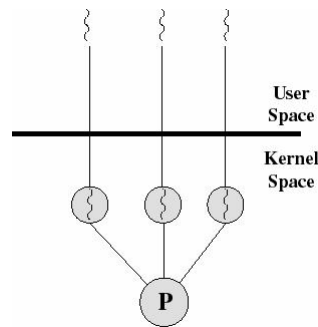
## YTIMEN SÄIKEET

KLT, Kernel Level Threads

28

## Säikeiden toteutus ytimessä

- Säikeiden hallinta kokonaan KJ:n ytimessä
- Toteutus ei käytä kirjastoa
  - ohjelmoijalle näkyvä rajapinta voi olla silti sama kuin edellä
  - palvelupyynnöt
- Ydin tietää säikeistä
  - prosesseista PCB:t
  - niiden säikeistä TCB:t
- Vuorottaminen KJ:n heiniä
  - aikaviipale säikeelle



29

## Säikeiden toteutus ytimessä

### Hyötyjä

- Prosessin säikeitä voi olla yhtäaikaan suorituksessa eri prosessoreilla
- Jos säie Blocked- tilaan, muut prosessin säikeet voivat silti jatkaa
- Myös KJ-ytimen toteutus voi käyttää säikeitä

### Haittoja

- Säikeen vaihto vaatii aina 2 vaihetta:
  - keskeytys + siirtyminen KJ:hin (-> etuoik. tila)
  - vuorottaminen (->kjätila)
- Vaihto hitaampaa kuin kirjastoa käytettäessä
  - ks. taulukko 4.1

Operation	User-Level Threads	Kernel-Level Threads	Processes
Null Fork	34	948	11,300
Signal Wait	37	441	1,840

30

## KLT-säikeen vs Prosessin tila

- KJ tietää säikeiden olemassaolosta
  - KJ ylläpitää säiekuvaajia
  - KJ vuorottaa säietasolla
    - jos prosessin aikaviipaleta jäljellä, vuorota saman prosessin säikeitä tai
    - koko aikaviipale säikeelle
- Jos KLT-säie odottaa palvelupyynnössä, prosessin muut säikeet voivat silti jatkaa

31

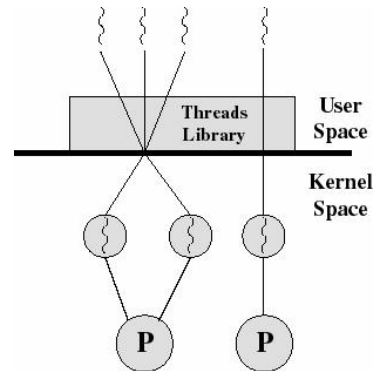
## KULTAINEN KESKITIE ?

32



## Solaris

- Yhdistää molempien parhaat piirteet
- Luonti käyttäjätilassa
- Synkronointi käyttäjätilassa
- Pääosa vuorottamisesta käyttäjätilassa
- Ohjelmoija voi *liittää* käyttäjätason säikeet ytimen säikeiksi haluamallaan tavalla



33

## Symmetric MultiProcessors (SMP)

34

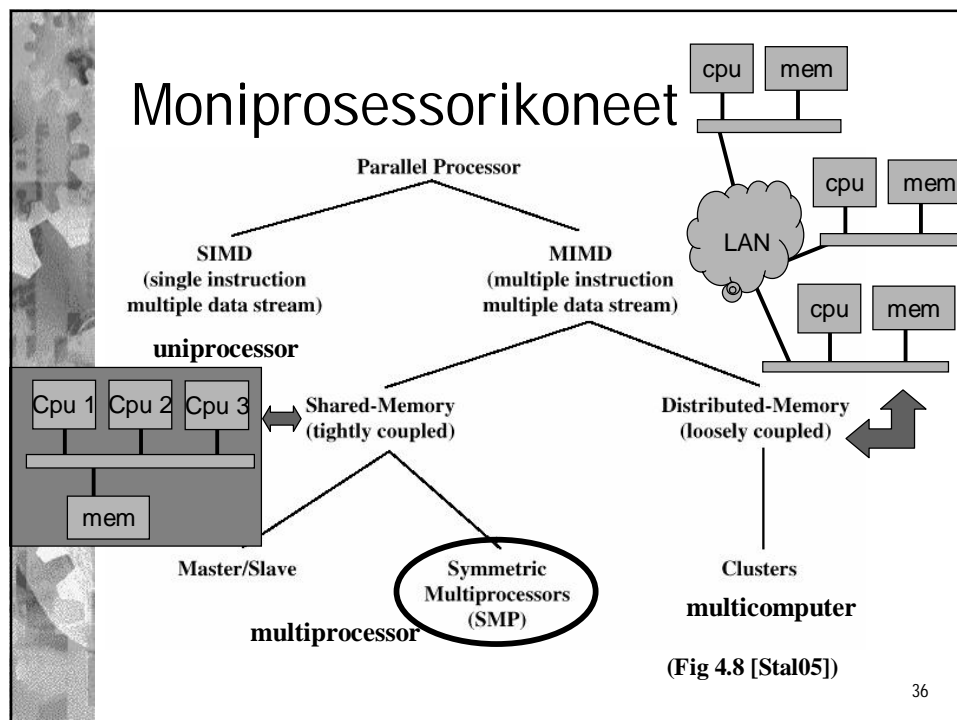
## Flynnin jaottelu moniprosessorikoneille

Michael J. Flynn, 1966

- SISD: Single Instruction - Single Data
  - yksi käsky, yksi data
  - normaali CPU: yksi ALU, yksi CU, yksi MEM
- SIMD: Single Instruction - Multiple Data
  - yksi käsky, monta data-alkiota
  - taulukkoprosessori: monta erikoistunutta ALUa
  - vektorisuoritin: vektorikonekäskyt, vektorirekisterit
- MISD: Multiple Instruction - Single Data
  - Ei mielekäs
- MIMD: Multiple Instruction - Multiple Data
  - monta käskyä ja monta dataa
  - tiukasti kytketty: monta CPU:ta, yhteinen MEM
  - löyhästi kytketty: monta CPU:ta, kullakin oma MEM

35

## Moniprosessorikoneet



36

## Master-slave vs. SMP

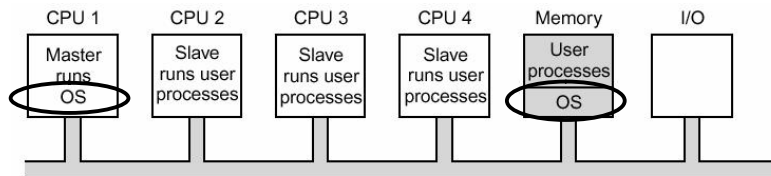


Fig. 8-8. A master-slave multiprocessor model. [Tane 01]

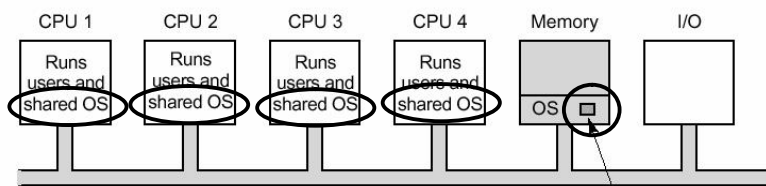
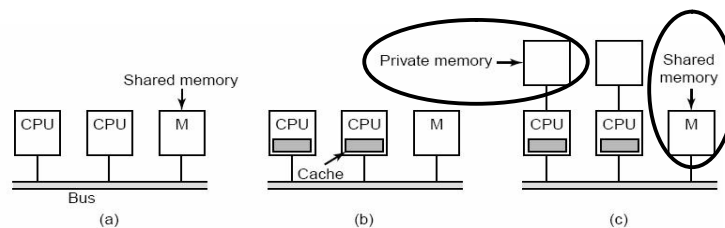


Fig. 8-9. The SMP multiprocessor model. [Tane 01]

37

## Toinen jaottelu yhteisen muistin moniprosessorikoneille



**(NUMA)**

(non-uniform memory arch)

Fig. 8-2. Three bus-based multiprocessors. (a) Without caching. (b) With caching. (c) With caching and private memories.

[Tane 01]

38

## Kolmas jaottelu yhteisen muistin moniprosessorikoneille

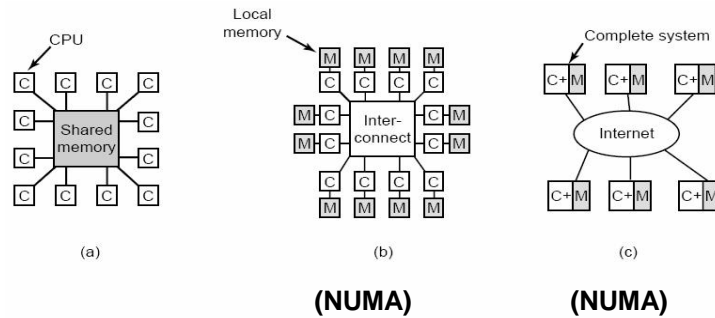
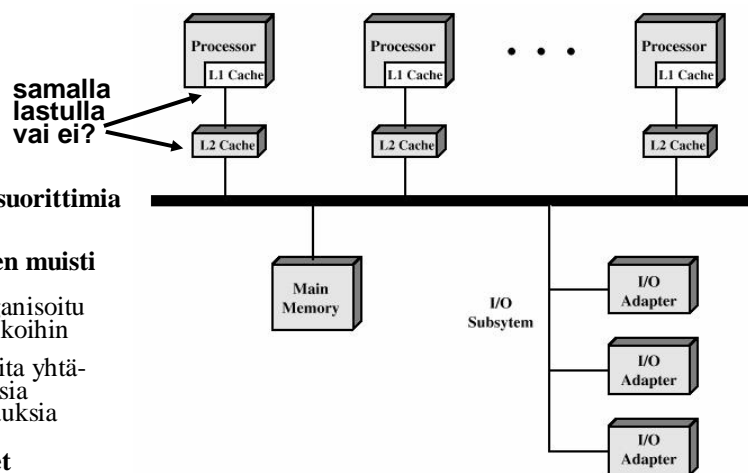


Fig. 8-1. (a) A shared-memory multiprocessor. (b) A message-passing multicomputer. (c) A wide area distributed system.

[Tane 01]

39

## SMP: Symmetric Multiprocessing



- Useita suorittimia (CPU)
- Yhteinen muisti
  - organisoitu lohkoihin
  - useita yhtäaikaisia viittauksia
- Yhteiset oheslaitteet
- Välimuistien yhteneväisyys (coherency)?

(Fig. 4.9 [Stal 05])

40

## Huomioitava SMP KJ:ssä 1/2

- Samanaikaisuuden hallinta (Ch 5-6, RIO)
  - KJ voi olla suoritettavana usealla CPU:lla
  - KJ:n oltava vapaakäyntinen (re-entrant)
  - KJ:n tietorakenteiden käsittelyssä tarvitaan poissulkemista
- Vuorotettavana useita CPU:ita (Ch 10)
  - KJ:kin voi olla suorituksessa yhtäaikaan eri CPU:illa
  - saman prosessin säikeiden vuorottaminen
- Synkronointi, lukot (Ch 5-6, RIO)
  - poissulkeminen ja tapahtumien järjestys

41

## Huomioitava SMP KJ:ssä 2/2

- Muistinhallinta (Ch 7-8)
- Ulkoisten keskeytysten käsittely
- Vikasietoisuus: joku CPU voi pudota pois
  - tästä ei tarvinnut murehtia yhden CPU:n kanssa!
  - ollaanko vikasietoisia vai ei? miten?

42