

Luento 8

Ohjelman toteutus järjestelmässä



Prosessi
Prosessin esitysmuoto
järjestelmässä
Käyttöjärjestelmä
KJ-prosessit

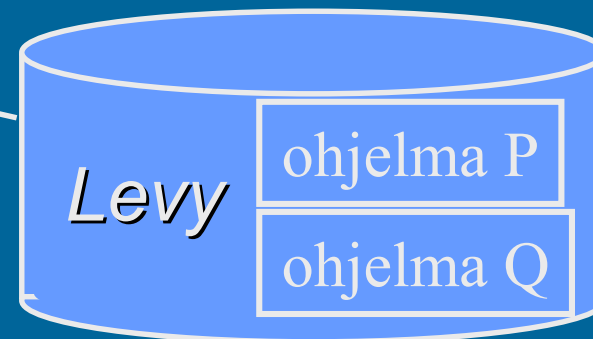
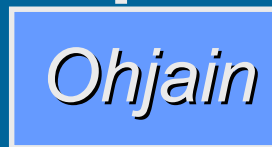
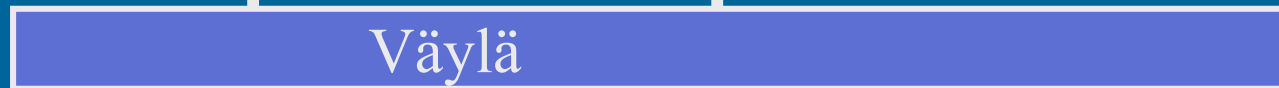
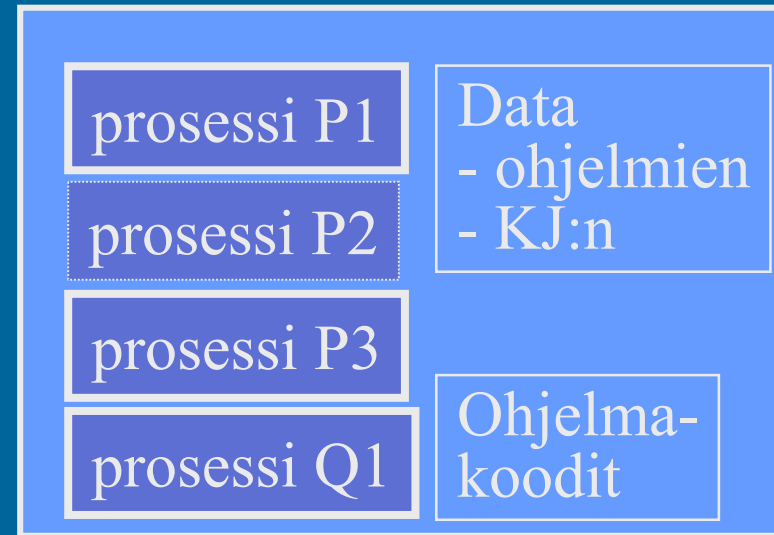
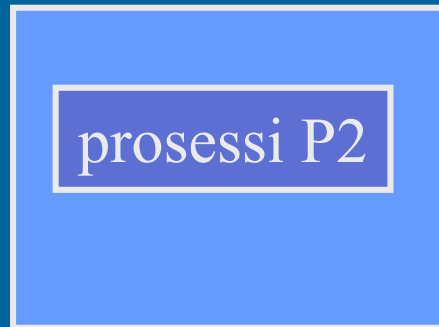
Prosessi ⁽⁴⁾

- Järjestelmässä olevan ohjelman esitysmuoto
- Järjestelmässä voi olla ”samalla kertaa” monta prosessia joko samasta tai eri ohjelmasta
 - käyttäjän (ihmisen) näkökulma ja aikaskaala (1 min, 1 sek?)
- Suorittimella suorituksessa on yksi prosessi kerrallaan
 - laitteiston näkökulma ja aikaskaala (1 ms, 1 μ s, 1 ns?)
- Muut prosessit ovat odottamassa jotakin
 - suoritinta? I/O:ta? viestiä toiselta prosessilta?
 - vapaata muistitilaa?

Prosessi

Muisti

Suoritin



Prosessin vaihto ⁽⁴⁾

- Suorittimella suoritusvuorossa olevan prosessin vaihtaminen
- Tapahtuu aika usein
 - keskimäärin noin 2000-3000 konekäskyn välein?
 - esim. 50-500 kertaa sekunnissa?
- Iso operaatio - paljon kopiointia
 - montako konekäskyä tähän kuluu?

50-500?

0?

Ks. Minix esimerkin `tty_int` [Tane87], 3 kalvoa

Prosessin elinkaari (11)



- Prosessin 5 suoritusilaa
 - Milloin tilanvaihto tapahtuu?
 - Mitä tilanvaihdossa tapahtuu?

Mitä suoritin tietää suorituksessa olevasta prosessista?

Prosessin esitysmuoto järjestelmässä ⁽⁵⁾

- PCB - Prosessin kuvaaja eli kontrollilohko (Process Control Block)
 - isohko tietue, joka sisältää kaiken yhdestä prosessista
 - muistialueet, tiedostot, tiedostojen käsittelykohdat
 - ei suorituksessa oleville myös: suorittimen tila (laiterekisterit, MMU:n rekisterit, kontrollirekisterit)
 - joka prosessista oma PCB
 - luodaan prosessin luonnin yhteydessä ja tuhotaan prosessin päättyessä
 - käyttöjärjestelmärutiinit manipuloivat PCB:tä

Prosessin kuvaajan sisältö ⁽⁹⁾

- Prosessin tunniste 14023
- Prioriteetti suorittimen vuoronantoa varten 143
- Prosessin tila ja/tai odottamisen syy R-to-R
- Suoritinympäristö talletettuna odottamisen aikana
 - rekisterit, PC, SP, FP, tilarekisterit
- Seuraavaksi suoritettavan käskyn osoite Main { }
- Poikkeuskäsittelijöiden osoitteet (ellei oletusarv.)
- Aikaviipale
- Käytössä olevat muistialueet, aukiolevat tiedostot
- KJ:n hallintotietoa (kokonaisaika, etc etc)

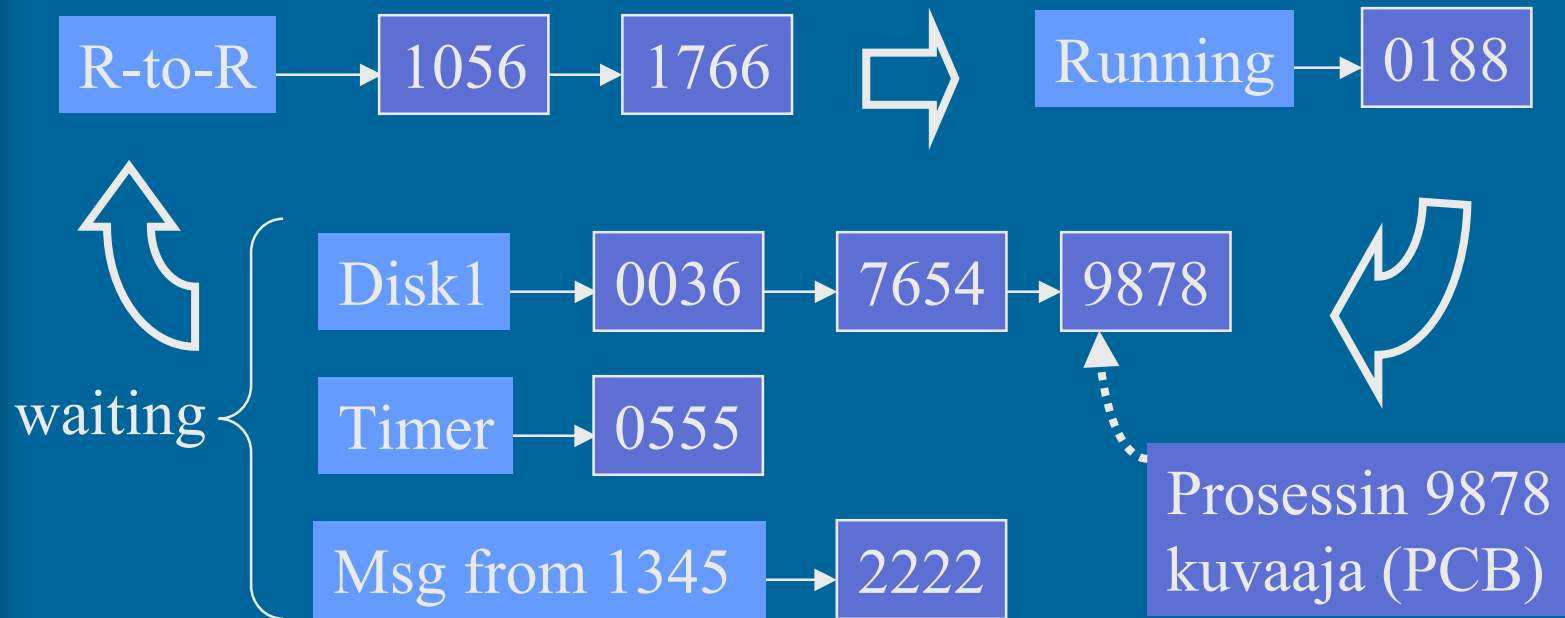
Ks. Minix esimerkin *struct proc* [Tane87], 1 kalvo

Prosessin tilanvaihdon toteutus (11)

- Prosessin tilanvaihto tapahtuu siirtämällä prosessi (sen PCB) jonosta toiseen
 - ready-to-run jono (tai jonot) odottaa suoritinta
 - running jono suorituksessa
 - ei oikeastaan ole olemassa
 - waiting jono odottaa jotakin
 - joka tyypille oma jononsa
 - esim: laitteen Disk1 I/O:n valmistumista odottavat
 - esim: näppäimistön painallusta odottavat
 - esim: kellolaitekeskeytystä odottavat
 - esim: prosessilta 1345 signaalia odottavat

Ks. Minix esimerkin *ready* [Tane87], 1 kalvo

Prosessit jonoissa (1)

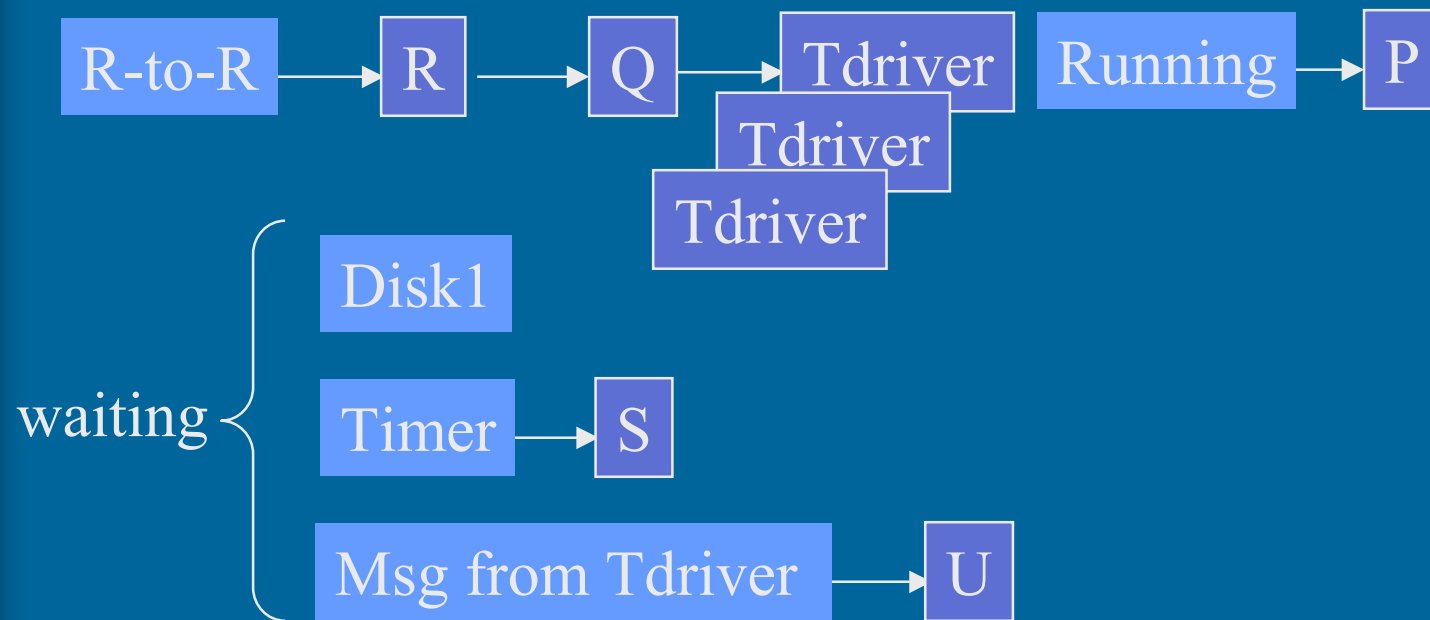


Vuoronanto:

valitse seuraava prosessi Ready-to-Run -jonosta ja siirrä se suoritukseen CPU:lle

(kopioi tämän prosessin **suorittimen tila** suorittimelle)

KJ esimerkki: I/O keskeytys (5)



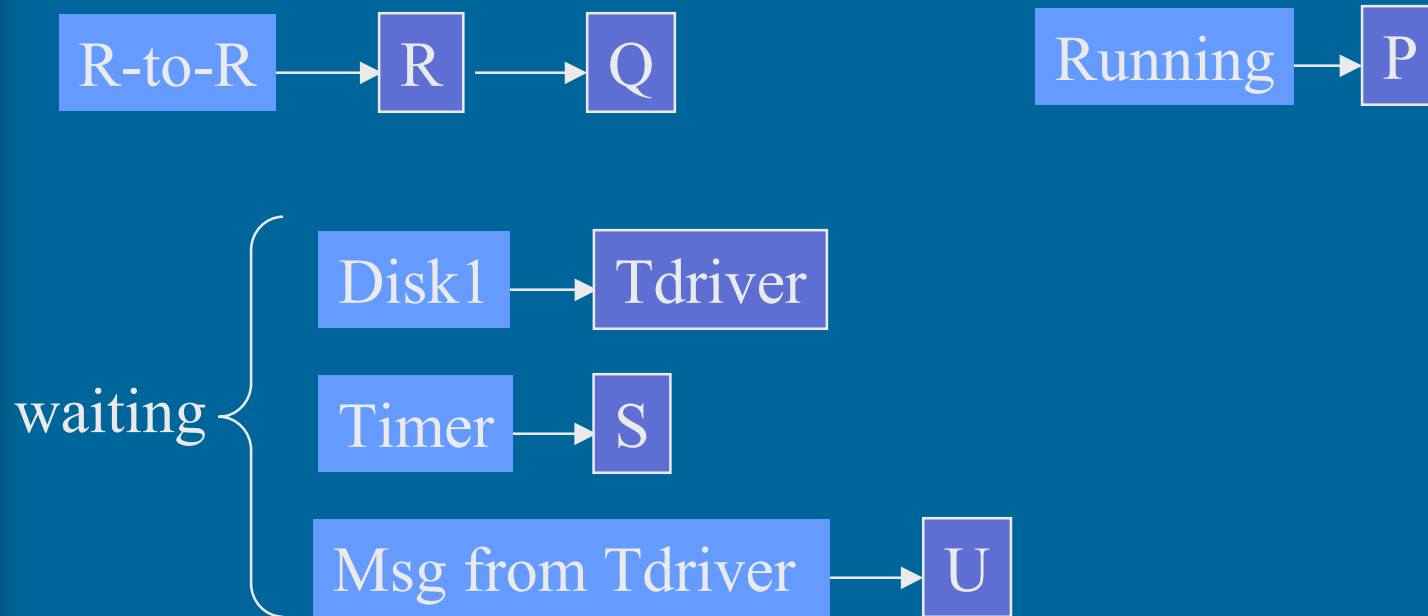
I/O keskeytys laitteelta Disk1 prosessille Tdriver?

Suoritin havaitsee keskeytyssignaalin ja suorittaa I/O keskeytyksittelyrutiinin (P:n ympäristössä)

Tdriver siirretään R-to-R jonoon

P:n suoritus jatkuu vai jatkuuko?

KJ esimerkki: I/O keskeytys (ei anim)

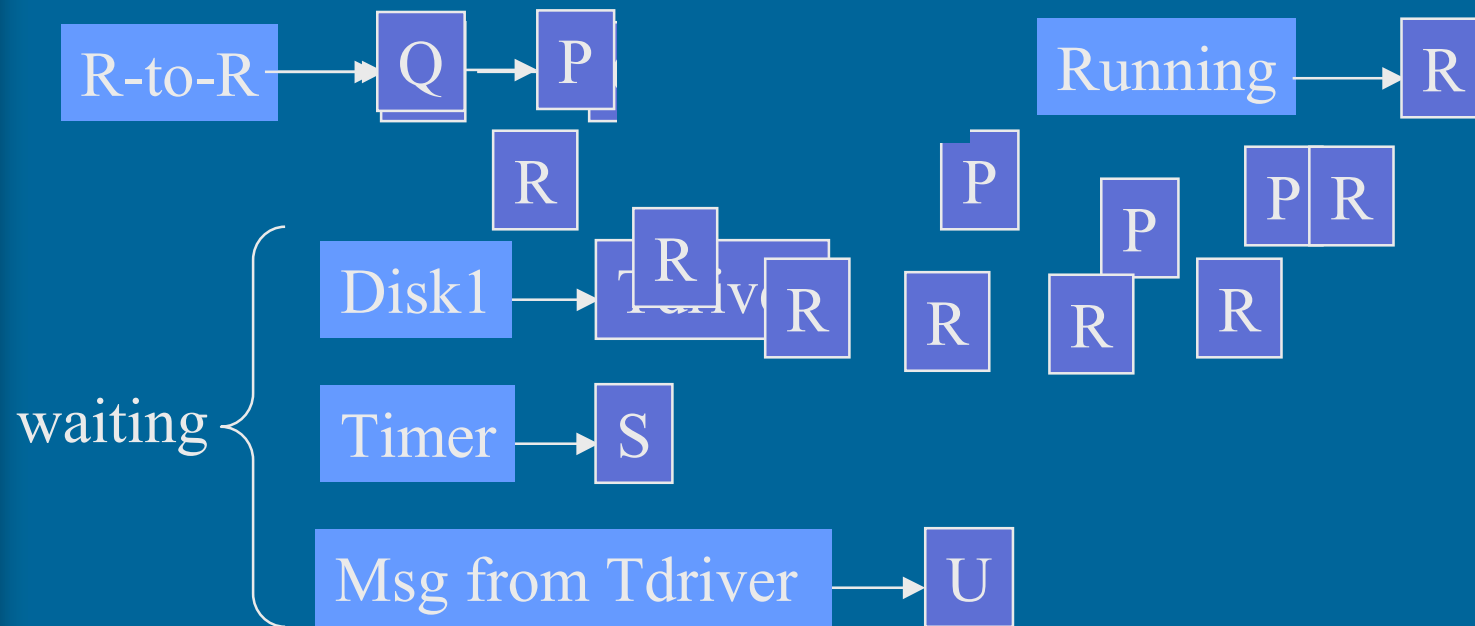


I/O keskeytys laitteelta Disk1 prosessille Tdriver?
Suoritin havaitsee keskeytyssignaalin ja suorittaa I/O keskeytyksittelyrutiinin (P:n ympäristössä)

Tdriver siirretään R-to-R jonoon

P:n suoritus jatkuu vai jatkuuko?

KJ esim: aikaviipalekeskeytys (7)



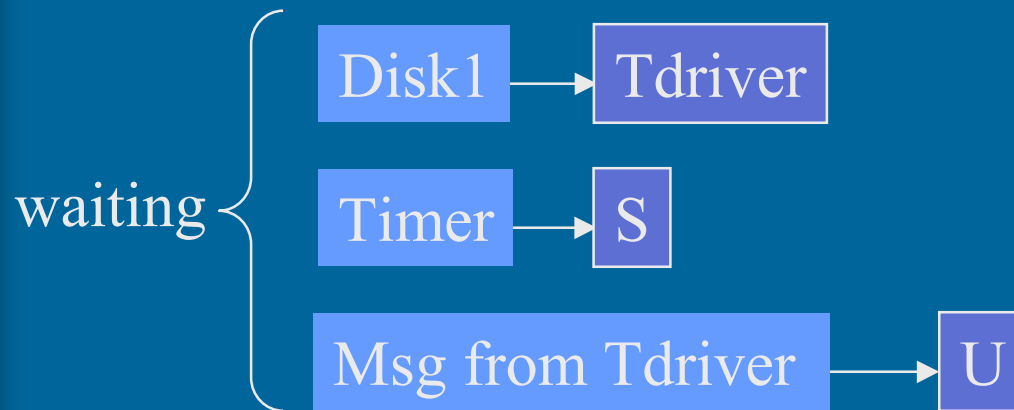
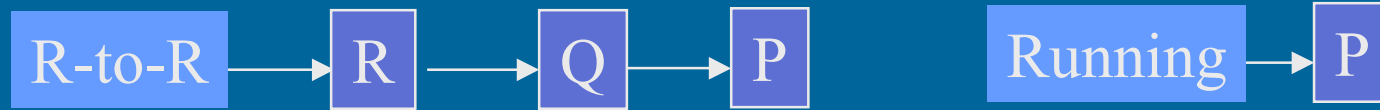
P saa aikaviipalekeskeytyksen?

P siirretään takaisin R-to-R jonoon

Seuraava prosessi R saa suoritusvuoron

Entä jos P olisi pyytänyt levy I/O:ta Disk1:ltä?

KJ esim: aikaviipalekesk. (ei anim)



P saa aikaviipalekeskeytyksen?

P siirretään takaisin R-to-R jonoon

Seuraava prosessi R saa suoritusvuoron

Entä jos P olisi pyytänyt levy I/O:ta Disk1:ltä?

Prosessin vaihto ⁽⁴⁾

- Vaihdon tekee KJ rutiini sillä hetkellä suorittavan prosessin ympäristössä
- Talleta vanhan prosessin suoritinympäristö suorittimelta omalle talletusalueelle muistiin
 - talleta kaikki suorittimella olevat tiedot muistiin
- Kopio uuden prosessin suoritinympäristö omalta talletusalueeltaan suorittimelle
 - lataa kaikki suorittimen rekisterit (myös PC!)
- Uuden prosessin suoritus jatkuu täsmälleen siitä mihin viime kerralla jäätiin
 - sama konekäsky, käytännössä sama suoritusympäristö

Ks. Minix esimerkin `tty_int` [Tane87], kalvo INT-3

- usein keskellä prosessin vaihtoa suorittavaa KJ rutiinia

Prosessin prioriteetti (3)

- Prosessin tärkeysjärjestys suorittimella
 - esim. pieni numero \Rightarrow iso (parempi) prioriteetti
- Joka prioriteetti(luokalle) oma R-to-R jononsa
 - KJ prosesseilla parempi prioriteetti kuin käyttäjätason prosesseilla
 - tosiaikasovelluksen prosesseilla parempi prioriteetti kuin KJ prosesseilla
 - muistakaa antaa KJ:lle aikaa aina joskus !
- Prioriteetti voi vaihdella prosessin elinaikana
 - paljon suoritinaikaa \Rightarrow huonompi prioriteetti
 - kauan R-to-R jonossa \Rightarrow parempi prioriteetti (prosessi siirretään korkeamman prioriteetin R-to-R jonoon)

Käyttäjän näkökulma (käyttö)järjestelmään ⁽⁵⁾

- Miten järjestelmä toimii minun ohjelmani kanssa?
- Onko järjestelmä riittävän nopea pelaamaan suosikkipeleäni isolla näytöllä?
- Onko minun helppo asentaa uusi ohjelma koneelle?
- Onko minun helppo muuntaa (portata) ohjelmani tähän käyttöjärjestelmään?
- Miten muistin lisääminen vaikuttaisi minun ohjelmani nopeuteen?

Käyttöjärjestelmän näkökulma (käyttö)järjestelmään ⁽⁶⁾

- Ovatko kaikki systeemin resurssit mahdollisimman hyvässä käytössä?
- Mikä on keskimääräinen jonon pituus (prosessien lukumäärä) suorittimelle?
- Minkä osan ajasta suoritin odottaa järkevää työtä?
- Minkä osan ajasta kovalevyn hakuvarsi on liikkessa?
- Miten usein datamuistiviitteet löytyivät välimuistista?
- Miten muistin lisääminen vaikuttaisi nopeuteen?

Käyttöjärjestelmä käyttöliittymänä laitteistoon ⁽³⁾

- Loppukäyttäjälle (ihmiselle)
- Sovellusohjelmille
- Piilottaa laitteiston erityispiirteet käyttäjiltä
 - käskykanta
 - konekäskäskyn rakenne
 - suorittimen toteutus ja suorittimien lukumäärä
 - I/O:n toteutus
 - I/O-laitteiden sijainti

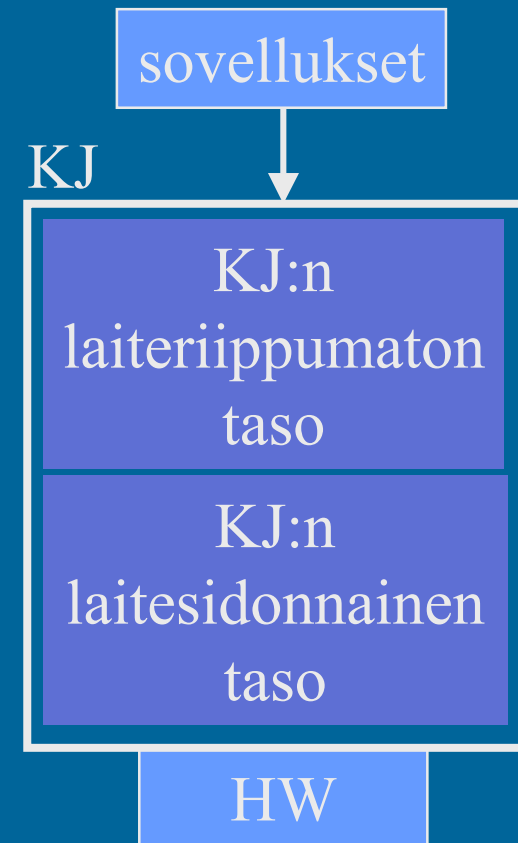
käyttäjät / sovellukset

KJ

HW

Käyttöjärjestelmän tavoitteet

- Laiteriippumaton (HW-riippumaton) käyttöliittymä laitteistoon
 - järjestelmää on helppo käyttää
 - järjestelmä antaa reilua palvelua kaikille
 - sovellukset on helppo tehdä
 - sovellukset on helppo siirtää muista järjestelmistä



Käyttöjärjestelmän tavoitteet (jatk)

- Järjestelmän resurssien tehokas hallinta
 - kaikista resursseista saada maksimihyöty
 - joustava resurssien yhteiskäyttö
 - tiukka tietosuoja

Käyttöjärjestelmä resurssien vartijana ⁽⁵⁾

- Suoritinaikaa reilusti kaikille
 - kukaan ei odota suoritinta ikuisesti
 - kriittiset prosessit saavat ajoissa suoritinaikaa
- Tiedostojen (koodi, data) tehokas käyttö
 - laitteesta ja sijainnista riippumaton käyttö
 - helppo yhteiskäyttö ja samalla tietojen suojaus
- Tietoliikenneverkkojen käyttö
 - laiteriippumaton käyttö
 - helppo yhteiskäyttö ja samalla tietojen suojaus
- Hallintokirjanpito

KJ järjestelmän eheyden turvaajana ⁽³⁾

- Varauduttu kaikkiin mahdollisiin virheisiin
- Sovellusohjelmat eivät voi häiritä KJ:tä tai muita prosesseja
 - tahallaan (esim. tietokonevirukset)
 - vahingossa (yleisin tapaus)
- Järjestelmä ei lukkiudu tai ”kaadu”
 - KJ:n omat tietorakenteet ovat aina eheitä
 - sovellusohjelmat eivät voi koskea KJ:n tietorakenteisiin
 - sovellusohjelmat aina lopulta antavat vuoron KJ:lle

Käyttöjärjestelmän rakenne (4)

- Prosessien hallinta
 - prosessien luonti, tuhoaminen
 - prosessien välinen viestintä (IPC, Inter-Process Comm)
 - kenelle suoritinaikaa ja milloin?
- Muistin hallinta
 - miten keskusmuistia varataan eri prosessien käyttöön?
 - kunkin prosessin muistitilan hallinta
 - yhteiskäyttö ja tiedon suojaus
- Tiedostojen ja laitteiden hallinta
 - miten tiedostoja voidaan lukea/kirjoittaa?
 - yhteiskäyttö ja tiedon suojaus
- Verkon hallinta
 - miten kommunikoida muiden koneiden kanssa?

Käyttöjärjestelmän toteutus (5)

- Joukko prosesseja ja/tai aliohjelmia
 - prosessit elävät omaa elämäänsä (etuoikeutetussa tilassa eli root'ina?)
 - swapper (Unix) - muistinhallintaprosessi
 - init prosessi (Unix) - kaikkien käyttäjätason prosessien ”äiti”
 - laiteajurit
 - aliohjelmat suoritetaan sen hetkisen prosessin ympäristössä (etuoikeutetussa tilassa?)
 - keskeytyskäsitteijät
 - saavat kontrollin aina tarvittaessa
 - aliohjelmakutsut, SVC, viestit
 - ajastimet ja muut keskeytykset

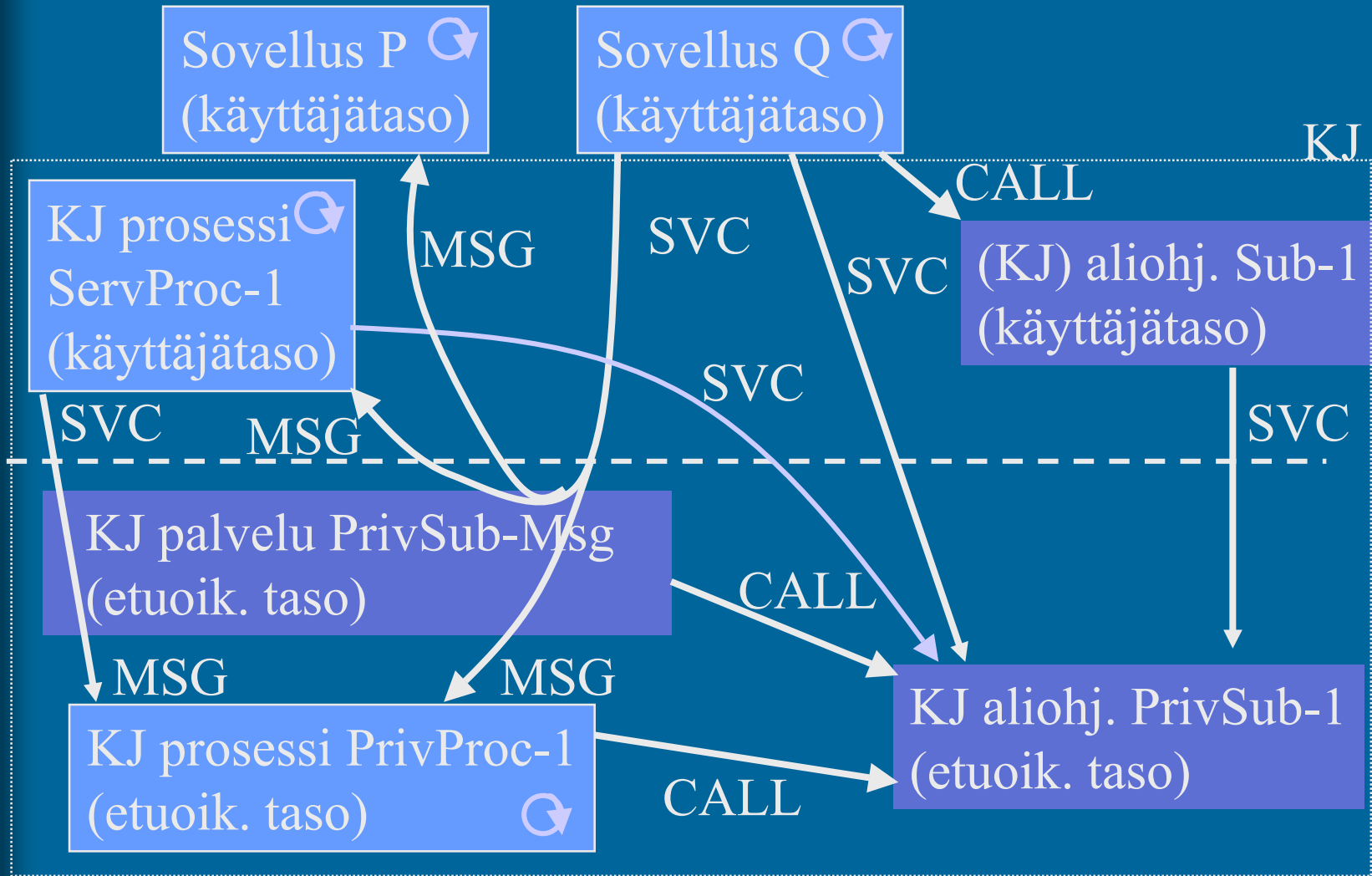
KJ palvelun kontrollin palautus

- Aliohjelmakutsut
 - CALL → RETURN
- SVC
 - SVC → IRET
- Viestit
 - viesti → vastausviesti
(lähettäjä odottaa vastausta RECEIVE:ssä)
- Ajastimet ja muut keskeytykset
 - keskeytys → IRET

Prosessit ja aliohjelmat (7)

suoritin
käyttäjätasolla

suoritin
etuoik.
tasolla

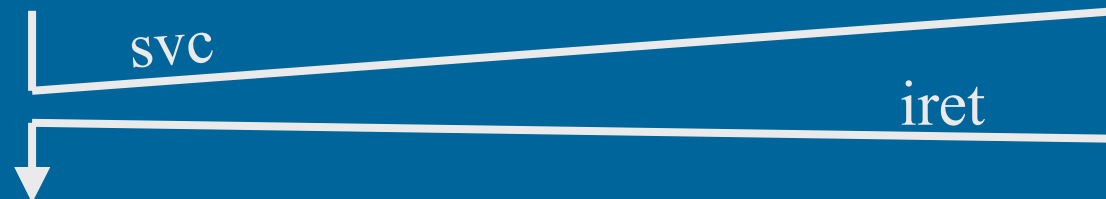


KJ esimerkki: laiteajuri

- Aliohjelmana (eli proseduurina)
 - laiteajuri suoritetaan KJ-rutiinina tavallisen SVC-kutsun kautta
 - vain yksi kutsu kerrallaan suorituksessa?
miksi? miten voidaan valvoa?

sov. prosessi

laiteajuri aliohj.

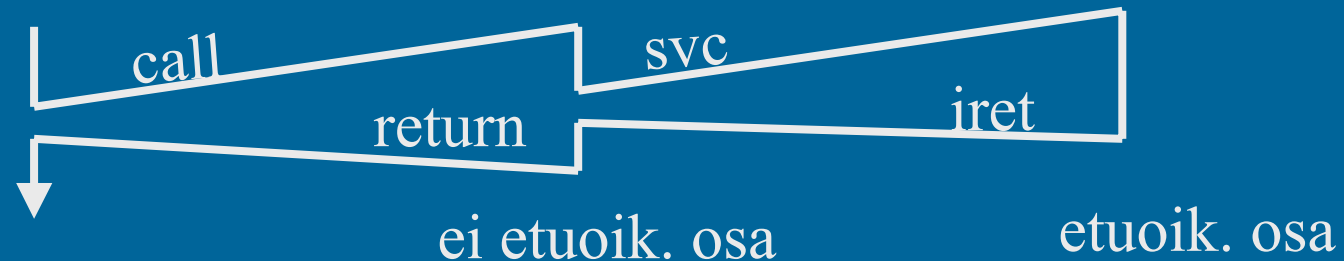


KJ esimerkki: laiteajuri (jatk.)

- Aliohjelmana (eli proseduurina)
 - laiteajuri suoritetaan KJ-rutiinina tavallisen aliohjelmakutsun ja/tai SVC-kutsun kautta
 - osa tai kaikki koodista voi olla etuoikeutettua
 - vain yksi kutsu kerrallaan suorituksessa?
miksi? miten voidaan valvoa?

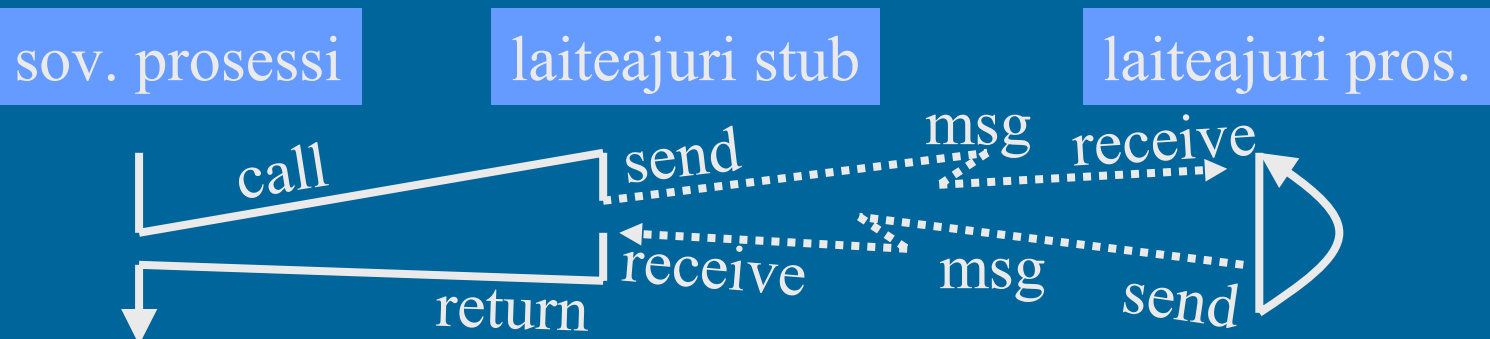
sov. prosessi

laiteajuri (osa etuoikeutettuna)



KJ esimerkki: laiteajuri

- Prosessina
 - proseduurina kutsuttu laiteajurin tynkä (stub) lähettää I/O-pyynnön viestinä laiteajuriprosessille ja odottaa vastausta
 - tynkä voi olla käyttäjätilainen
 - ajuriprosessi voi olla (joskus) etuoikeutettu
 - vaatii prosessien välistä viestintää



Ks. Minix esimerkki floppy disk driver [Tane87], 2 kalvoa

-- Luennon 8 loppu --

[Tane99]

```
// Open files for input and output.
inhandle = CreateFile("data", GENERIC_READ, 0, NULL, OPEN_EXISTING, 0, NULL);
outhandle = CreateFile("newf", GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
    FILE_ATTRIBUTE_NORMAL, NULL);

// Copy the file.
do {
    s = ReadFile(inhandle, buffer, BUF_SIZE, &count, NULL);
    if (s > 0 && count > 0) WriteFile(outhandle, buffer, count, &ocnt, NULL);
while (s > 0 && count > 0);

// Close the files.
CloseHandle(inhandle);
CloseHandle(outhandle);
```

Figure 6-40. A program fragment for copying a file using the Windows NT API functions. This fragment is in C because Java hides the low-level system calls and we are trying to expose them.

Lisää
tietoa?



KJ kurssit,
RIO, Hajjärj