Ohjelman ja käyttöjärjestelmän toteutus

- Prosessi
  - Ohjelman esitysmuoto
  - Prosessin toteutus järjestelmässä
- Käyttöjärjestelmä
  - Perustoiminnot
  - Käyttöjärjestelmäprosessit
  - Käyttöjärjestelmän toteutus
  - Kontrollin siirto prosessien välillä

Copyright Teemu Kerola 2005

Tällä luennolla käymme läpi prosessin käsitteen ja sen toteutuksen järjestelmässä. Käymme myös läpi käyttöjärjestelmän perustoiminnot ja perusrakenteet. Esittelemme myös peruspiirteissään, kuinka käyttöjärjestelmä toteutetaan prosessien ja aliohjelmien avulla. Tärkeänä osana käyttöjärjestelmän toteutusta on kontrollin siirto suorittavien moduulien välillä.

Prosessi

Järjestelmässä olevan ohjelman esitysmuoto

Järjestelmässä voi olla samalla kertaa monta prosessia samasta tai eri ohjelmasta

- käyttäjän (ihmisen) näkökulma ja aikaskaala (tunteja, minuutteja, sekunteja?)

Suorittimella suorituksessa vain yksi prosessi kerrallaan

- laitteiston näkökulma ja aikaskaala (millisekunteja, mikrosekunteja, nanosekunteja?)

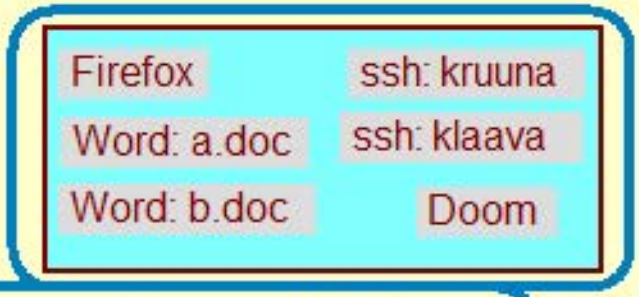
Muut prosessit ovat odottamassa jotain

- vuoroa suorittimelle?
- I/O:n päättymistä?
- viestiä joltain toiselta prosessilta (eri järjestelmästä)?
- vapaata muistitilaa?
- herätyskellon pärinää eli kellolaitekeskeytystä?

Copyright Teemu Kerola 2005

Prosessin käsite on perustavaa laatua oleva nerokas idea, johon oikeastaan kaikki tietokoneen toiminta pohjautuu. Tietokonejärjestelmän prosessilla tarkoitetaan yhden suorituksessa olevan ohjelman esitysmuotoa. Se on siis yksi selkeästi määritelty kokonaisuus, jonka avulla toteutetaan jonkin tietyn ohjelman yksi suorituskerta tässä järjestelmässä alusta loppuun. Tietokonejärjestelmän prosessia ei tule sekoittaa esimerkiksi puusta paperiksi -prosessiin, joka sinällään on selkeästi määritelty, mutta jonka suorittajien joukko on vaihtuva ja heterogeeninen. Tietokonejärjestelmän prosessi suoritetaan yhdessä laitteistossa yhden ohjelmakoodin perusteella.

Prosessi



Järjestelmässä olevan ohjelman esitysmuoto

Järjestelmässä voi olla samalla kertaa monta prosessia samasta tai eri ohjelmasta

- käyttäjän (ihmisen) näkökulma ja aikaskaala (tunteja, minuutteja, sekunteja?)

Suorittimella suorituksessa vain yksi prosessi kerrallaan

- laitteiston näkökulma ja aikaskaala (millisekunteja, mikrosekunteja, nanosekunteja?)

Muut prosessit ovat odottamassa jotain

- vuoroa suorittimelle?
- I/O:n päättymistä?
- viestiä joltain toiselta prosessilta (eri järjestelmästä)?
- vapaata muistitilaa?
- herätyskellon pärinää eli kellolaitekeskeytystä?

Copyright Teemu Kerola 2005

Samasta ohjelmasta voi järjestelmässä olla yhtäaikaa suorituksessa monta instanssia ja kutakin niitä vastaa oma prosessinsa. Kaikki järjestelmässä olevat prosessit näyttävät olevan yhtäaikaa suorituksessa, vaikka oikeasti niiden suoritusvuoroa vain vuorotellaan riittävän usein. Tämä ei ole sattuma, vaan normaalin pöytäkoneen tai läppärin käyttöjärjestelmä on nimenomaan suunniteltu tällä tavoin. Käyttäjän eli ihmisen näkökulmasta on mukavaa, kun tietokonejärjestelmä tuntuu suorittavan koko ajan kaikkia sille annettuja tehtäviä. On mukavampaa saada koko ajan vähän heikompaa palvelua, kuin ensin odottaa ehkä kauankin aikaa ja sitten saada palvelu nopeasti. Vai onko?

Prosessi

Järjestelmässä olevan ohjelman esitysmuoto

Järjestelmässä voi olla samalla kertaa monta prosessia samasta tai eri ohjelmasta

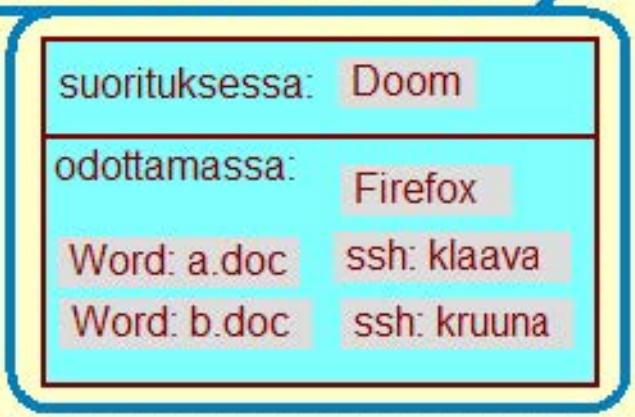
- käyttäjän (ihmisen) näkökulma ja aikaskaala (tunteja, minuutteja, sekunteja?)

Suorittimella suorituksessa vain yksi prosessi kerrallaan

- laitteiston näkökulma ja aikaskaala (millisekunteja, mikrosekunteja, nanosekunteja?)

Muut prosessit ovat odottamassa jotain

- vuoroa suorittimelle?
- I/O:n päättymistä?
- viestiä joltain toiselta prosessilta (eri järjestelmästä)?
- vapaata muistitilaa?
- herätyskellon pärinää eli kellolaitekeskeytystä?



Copyright Teemu Kerola 2005

Oikeasti suorituksessa on tietenkin vain yksi ohjelman instanssi eli prosessi kerrallaan. Onhan järjestelmässäkin vain yksi joukko rekistereitä ja esimerkiksi vain yksi paikanlaskuri (PC). Jos esimerkiksi kolmen ohjelman suoritusta eli siis kolmea prosessia vuorotellaan suorittimella aina 20 millisekunnin välein, niin ihminen ei mitenkään huomaa vuoronvaihtoja, vaan kaikki prosessit tuntuvat etenevät koko ajan. Todellisuudessa suoritin suorittaa kuitenkin vain yhtä prosessia kerrallaan aikaisemmin esitetyn käskyjen nouto- ja suoritussyklin mukaisesti. Vuoronvaihtoon kuluu tietenkin aina vähän aikaa, mutta ei kovin paljoa.

Prosessi

Järjestelmässä olevan ohjelman esitysmuoto

Järjestelmässä voi olla samalla kertaa monta prosessia samasta tai eri ohjelmasta

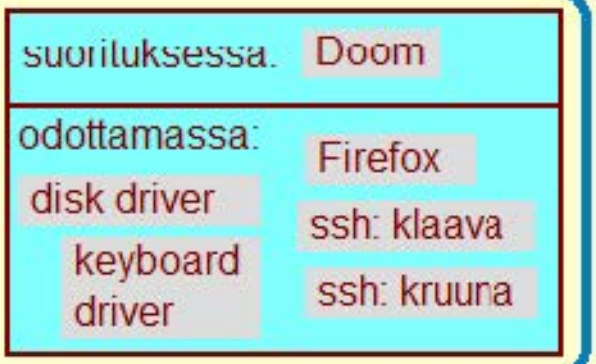
- käyttäjän (ihmisen) näkökulma ja aikaskaala (tunteja, minuutteja, sekunteja?)

Suorittimella suorituksessa vain yksi prosessi kerrallaan

- laitteiston näkökulma ja aikaskaala (millisekunteja, mikrosekunteja, nanosekunteja?)

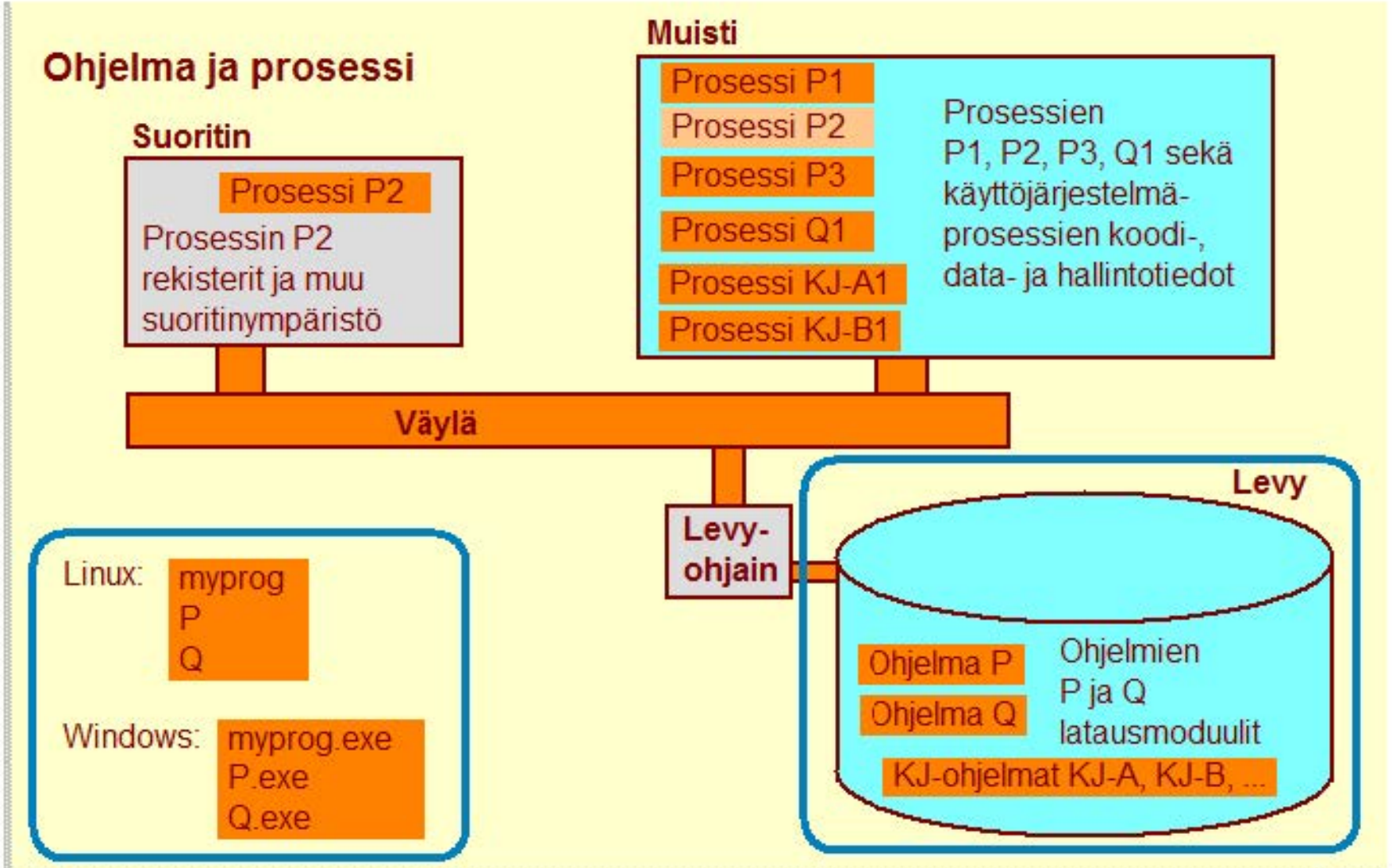
Muut prosessit ovat odottamassa jotain

- vuoroa suorittimelle?
- I/O:n päättymistä?
- viestiä joltain toiselta prosessilta (eri järjestelmästä)?
- vapaata muistitilaa?
- herätyskellon pärinää eli kellolaitekeskeytystä?



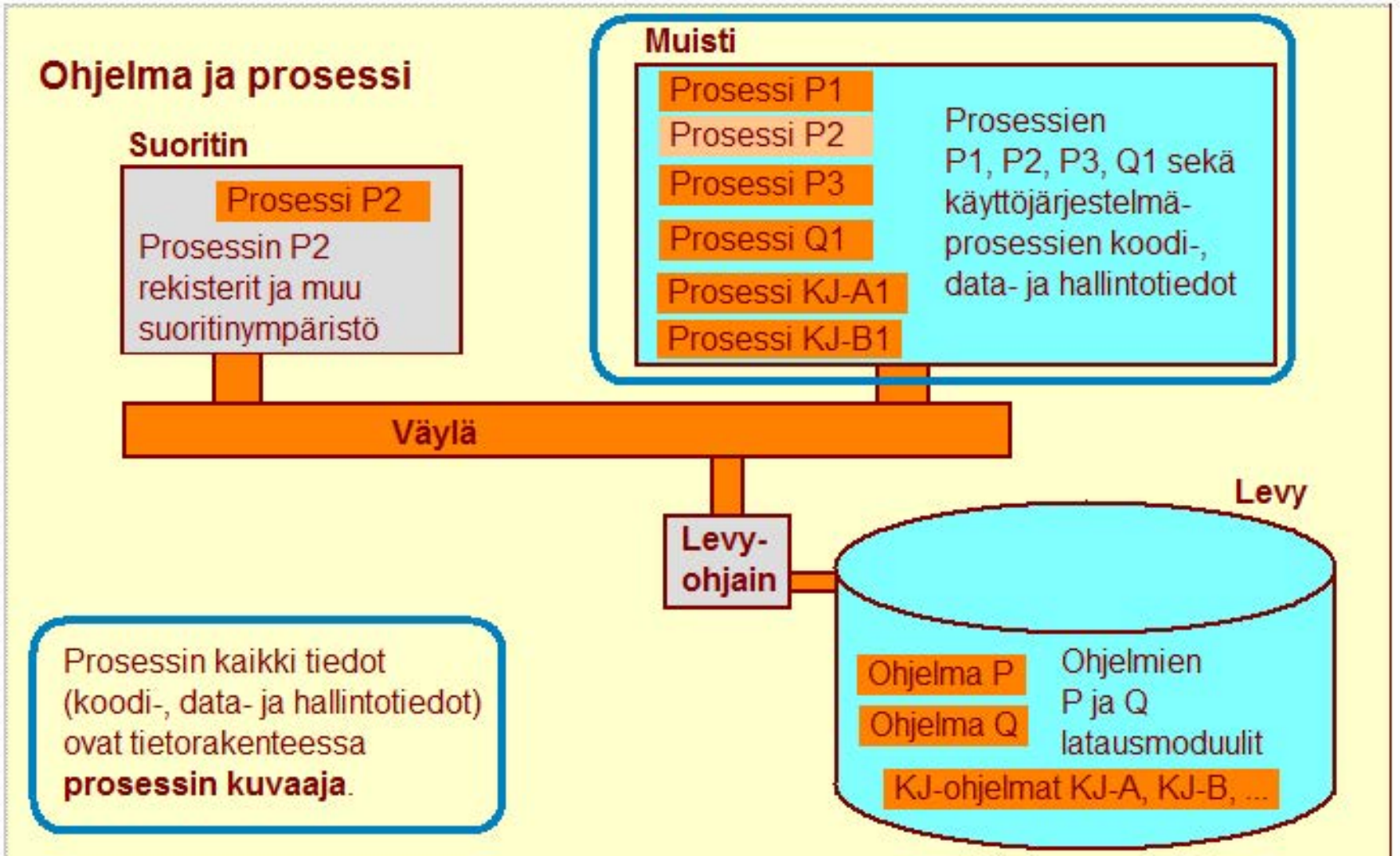
Copyright Teemu Kerola 2005

Kun yhtä prosessia suoritetaan, niin kaikki muut järjestelmässä olevat prosessit ovat odottamassa jotain. Osa niistä on valmiita suoritukseen ja odottaa vain suorittimelle pääsyä eli suoritusvuoroa, mutta yleensä suuri osa prosesseista odottaa jotain muuta. Prosessi voi esimerkiksi odottaa käyttäjältä I/O:ta hyvinkin kauan aikaa, koska I/O-laitteet (ja käyttäjät) ovat niin hitaita verrattuna suorittimen nopeuteen. Toisaalta taas esimerkiksi levymuistin toiminnasta vastaava laiteajuriprosessi voi olla odottamassa uutta työtä, toisin sanoen se odottaaa, kunnes jokin muu prosessi pyytää sitä tekemään levy-I/O:ta.



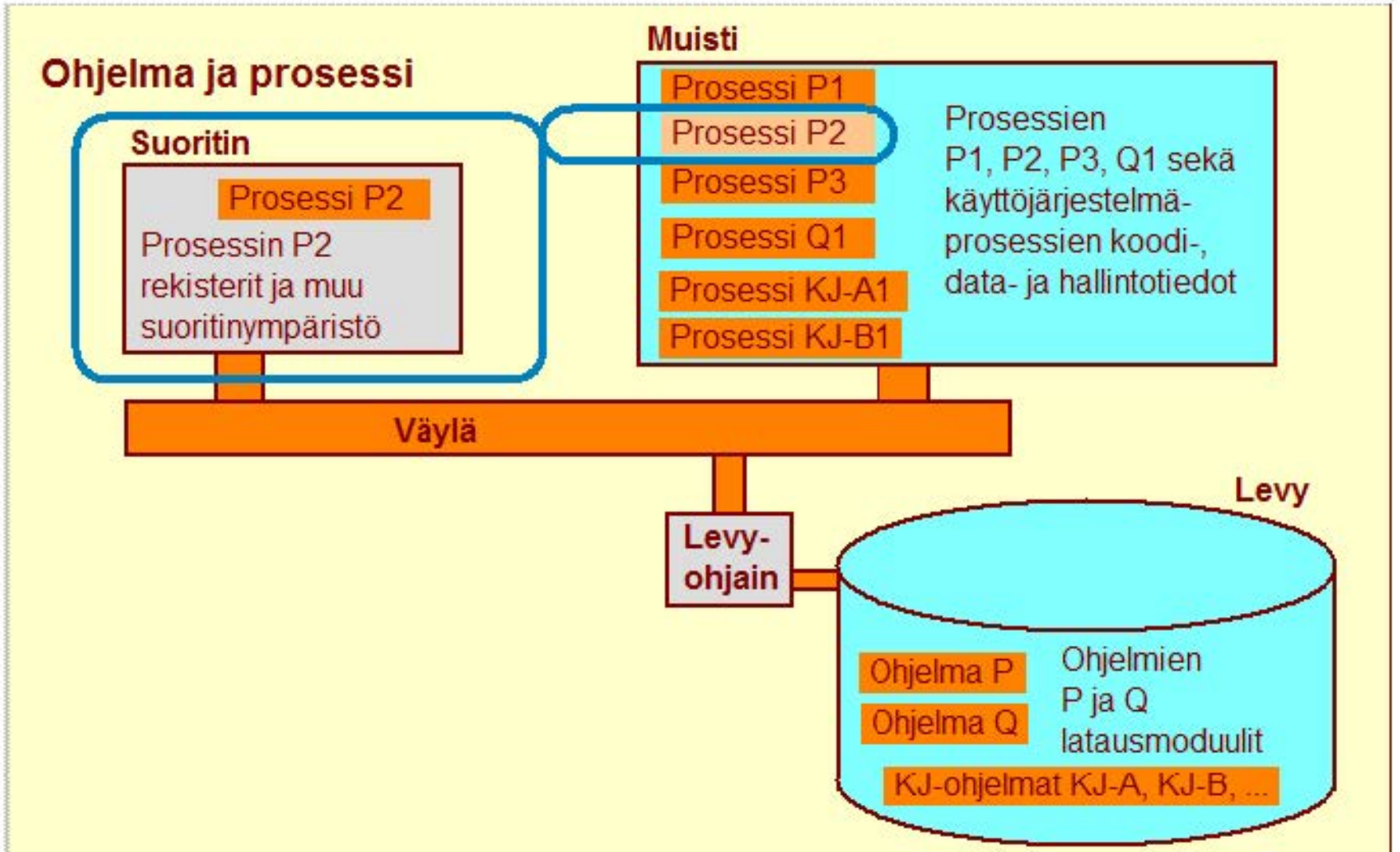
Copyright Teemu Kerola 2005

Kääntämisen ja linkityksen jälkeen ohjelmat talletetaan magneettisille tai optisille massamuistilaitteille. Pöytäkoneessa tämä on yleensä tavallinen kovalevy, mutta esimerkiksi Sonyn Playstation II:ssa se on yleensä DVD-levy. Ohjelma esitysmuoto on latausmoduuli, josta käyttöjärjestelmään kuuluva lataaja voi sitten luoda suorituskelpoisia prosesseja tarvittaessa. Latausmoduulit tunnistaa Windows-järjestelmissä loppuliitteestä '.exe'. Linux-järjestelmissä latausmoduuleilla ei ole loppuliitettä. Molemmissa järjestelmissä latausmoduulien alussa on selkeät hallintotiedot, josta myös ilmenee se, että kyseessä on latauskelpoinen moduuli.



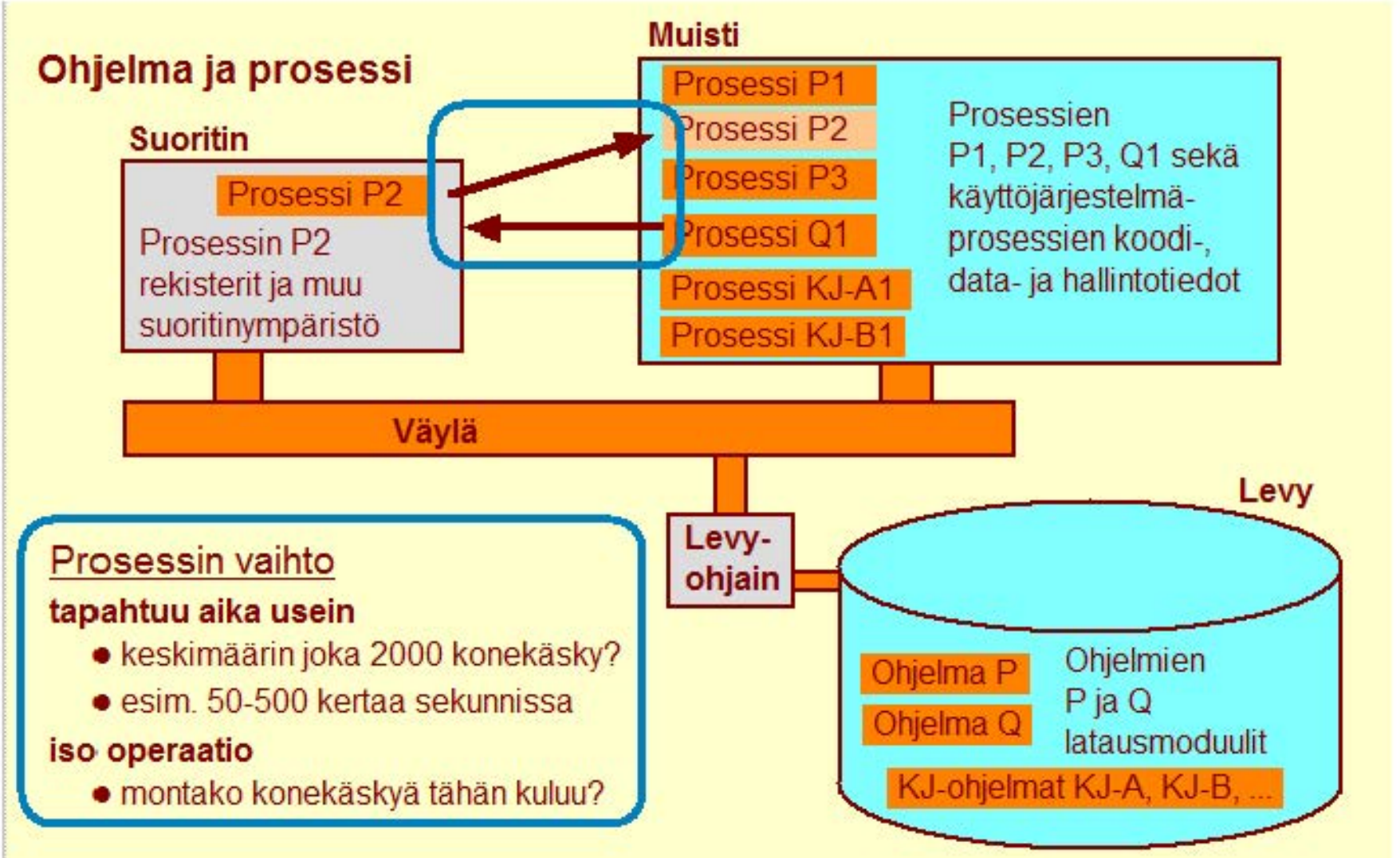
Copyright Teemu Kerola 2005

Latauksen jälkeen ohjelmasta on luotu siis järjestelmän tuntema prosessi, ja kunkin prosessin koodi- ja data-alueille on varattu tilaa muistista. Jos prosessilla on kaikki sen vaatimat resurssit, niin se on myös valmis suoritettavaksi suorittimella. Suorituskelpoisen prosessin kaikkien tietojen (koodi ja data) täytyy olla nimenomaan muistissa, koska ainoastaan muisti on tarpeeksi nopea laite suorittimen aikaskaalassa. Jokaiselle prosessille on omat tietonsa muistissa, vaikka tehokkuuden vuoksi jotkut niistä voivat olla yhteiskäyttöisiä. Esimerkiksi, prosesseilla P1, P2 ja P3 voi olla muistissa yhteinen koodisegmentti, koska ne ovat kaikki saman ohjelman P eri ilmentymiä.



Copyright Teemu Kerola 2005

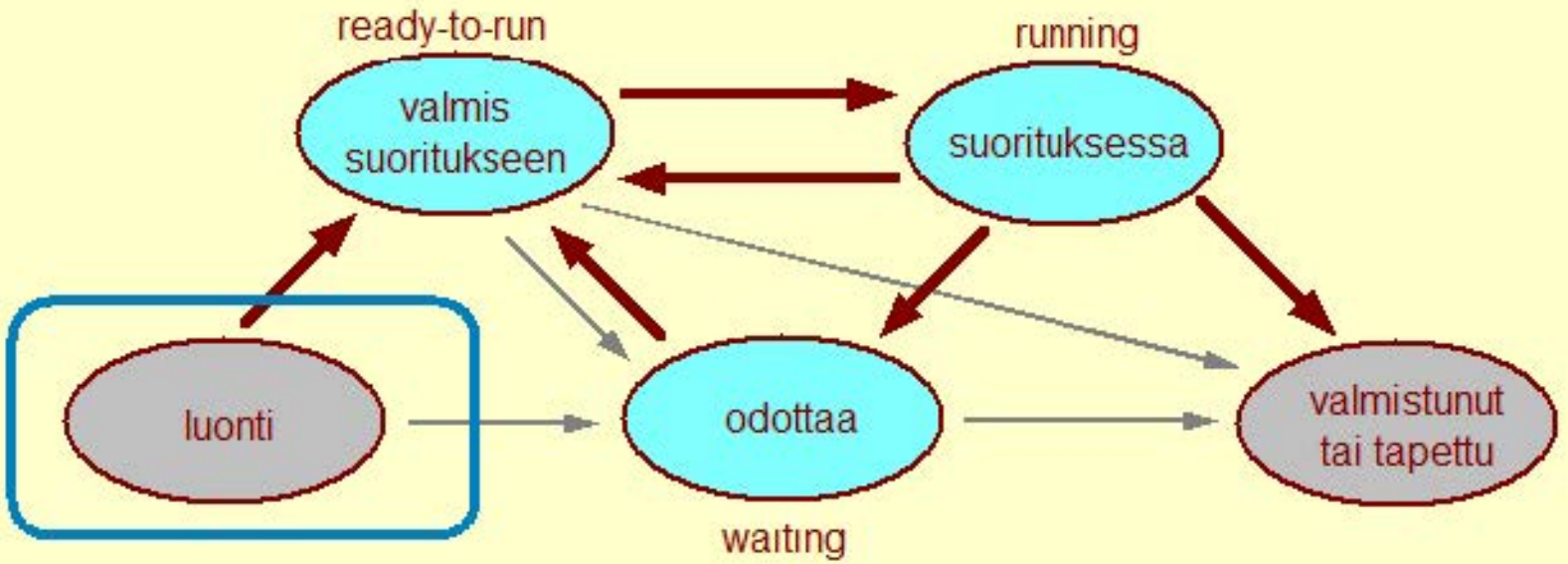
Suorittimella on siis yksi prosessi kerrallaan. Kaikki suorittimen rekisterit on prosessin vaihdon yhteydessä alustettu tälle prosessille, mutta pääosa prosessin tiedoista sijaitsee silti muistissa. Rekistereitä on vain muutama, vaikka prosessin data-alue voi olla mega-tavuja. Vastaavasti koodista on vain yksi konekäsky kerrallaan suorittimella ja muut muistissa. Prosessiin liittyvä hallintotieto on pääosin koko ajan muistissa käyttöjärjestelmän ylläpitämällä muistialueella.



Copyright Teemu Kerola 2005

Prosessin vaihdolla tarkoitetaan suorittimella suoritusvuorossa olevan prosessin vaihtamista. Aikaisemmin suoritusvuorossa olleen prosessin tiedot kopioidaan talteen muistiin ja suorittimen rekisterit alustetaan suoritusvuoron saaneen prosessin tiedoilla, jotka siis löytyvät muistista. Prosessin vaihto tapahtuu hyvin usein, mutta viimeistään silloin kun suoritusvuorossa oleva prosessi tekee jotakin, minkä vuoksi se ei enää voi jatkaa suoritusta. Tällaisia toimintoja ovat esimerkiksi datan lukeminen kovalevyltä tai käyttäjän syötteen vastaanotto näppäimistöltä. Prosessin vaihdon toteuttamiseen voi kulua esimerkiksi 50-500 konekäskyä.

Prosessin elinkaari



Prosessin viisi suoritustilaa

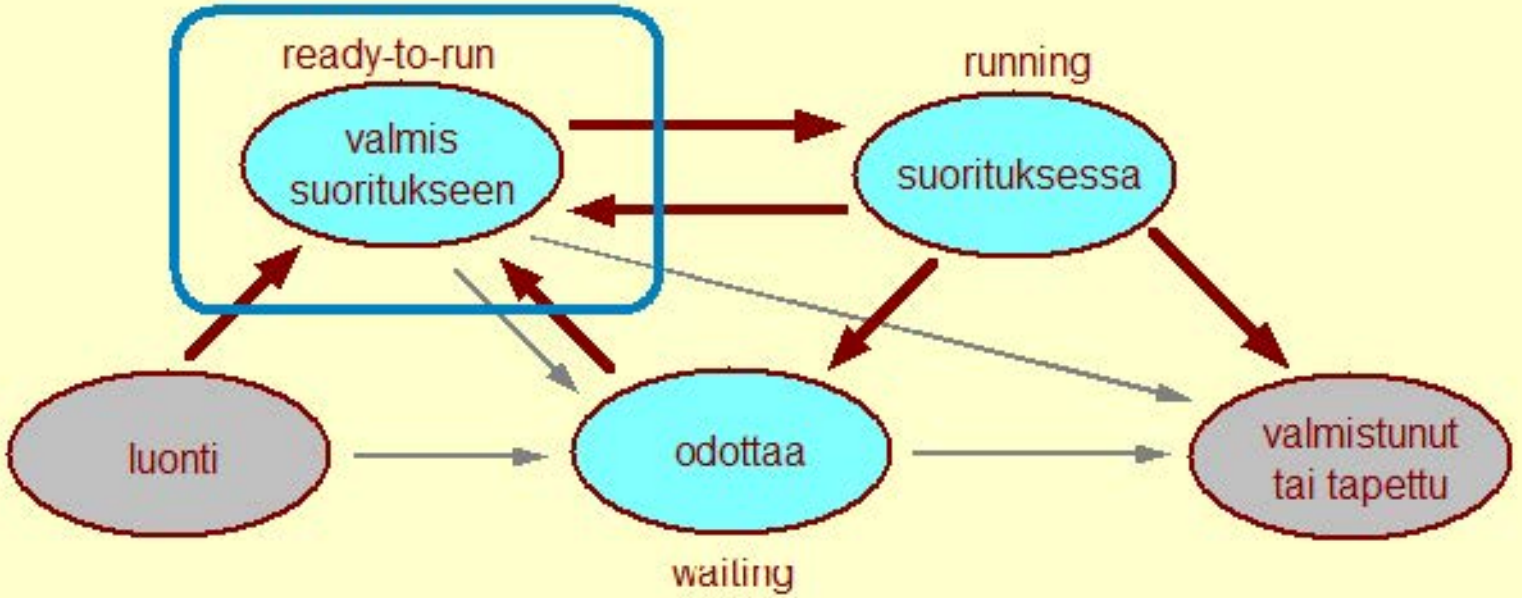
- mitä tila tarkoittaa?
- milloin tilanvaihto tapahtuu?
- mitä tilanvaihdossa tapahtuu?

Mitä suoritin tietää suorituksessa olevasta prosessista?

Copyright Teemu Kerola 2005

Prosessin elinkaari järjestelmässä alkaa, kun prosessi luodaan. Esimerkiksi Linux-komentotulkin tulkitsemasta ls-komennosta luodaan järjestelmään uusi prosessi, joka listaa joko oletusarvoisen tai parametrina annetun hakemiston tiedot. Käyttöjärjestelmä luo uudelle prosessille kaikki hallintorakenteet ja varaa sen tarvitsemat resurssit. Jos kaikki resurssit saacaan varattua, niin uusi prosessi pääsee heti jonottamaan suoritusvuoroa suorittimelle ready-to-run -jonoon. Muussa tapauksessa prosessi jää heti alkuun odottamaan puuttuvan resurssin, esimerkiksi muistitilan, vapautumista odotustilaan.

Prosessin elinkaari



Prosessin viisi suoritustilaa

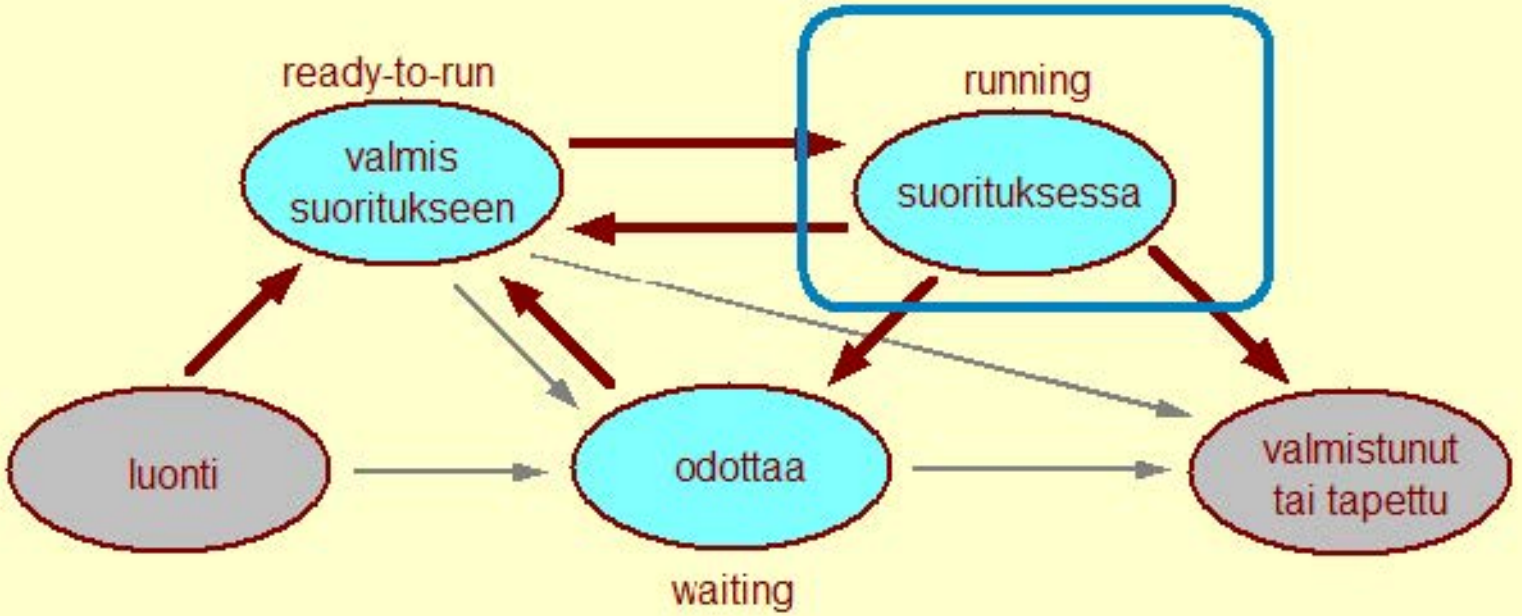
- mitä tila tarkoittaa?
- milloin tilanvaihto tapahtuu?
- mitä tilanvaihdossa tapahtuu?

Mitä suoritin tietää suorituksessa olevasta prosessista?

Copyright Teemu Kerola 2005

Valmis suoritukseen -tilassa ovat kaikki sellaiset prosessit, jotka vain odottavat suoritusvuoroa suorittimelle. Näille kaikille prosesseille on olennaista, että niiden kaikki koodi- ja data-alueet ovat valmiiksi käytettävissä muistissa. Tästä voi myös olla haittaa, jos ready-to-run -jonossa on paljon prosesseja, jotka kukin vievät paljon muistitilaa. Ääritapauksissa jonkin tällaisen prosessin tiedot voidaan joutua siirtämään levylle muistitilan vapauttamiseksi, jolloin tietenkin myös kyseinen prosessi siirretään odotustilaan odottamaan.

Prosessin elinkaari



Prosessin viisi suoritustilaa

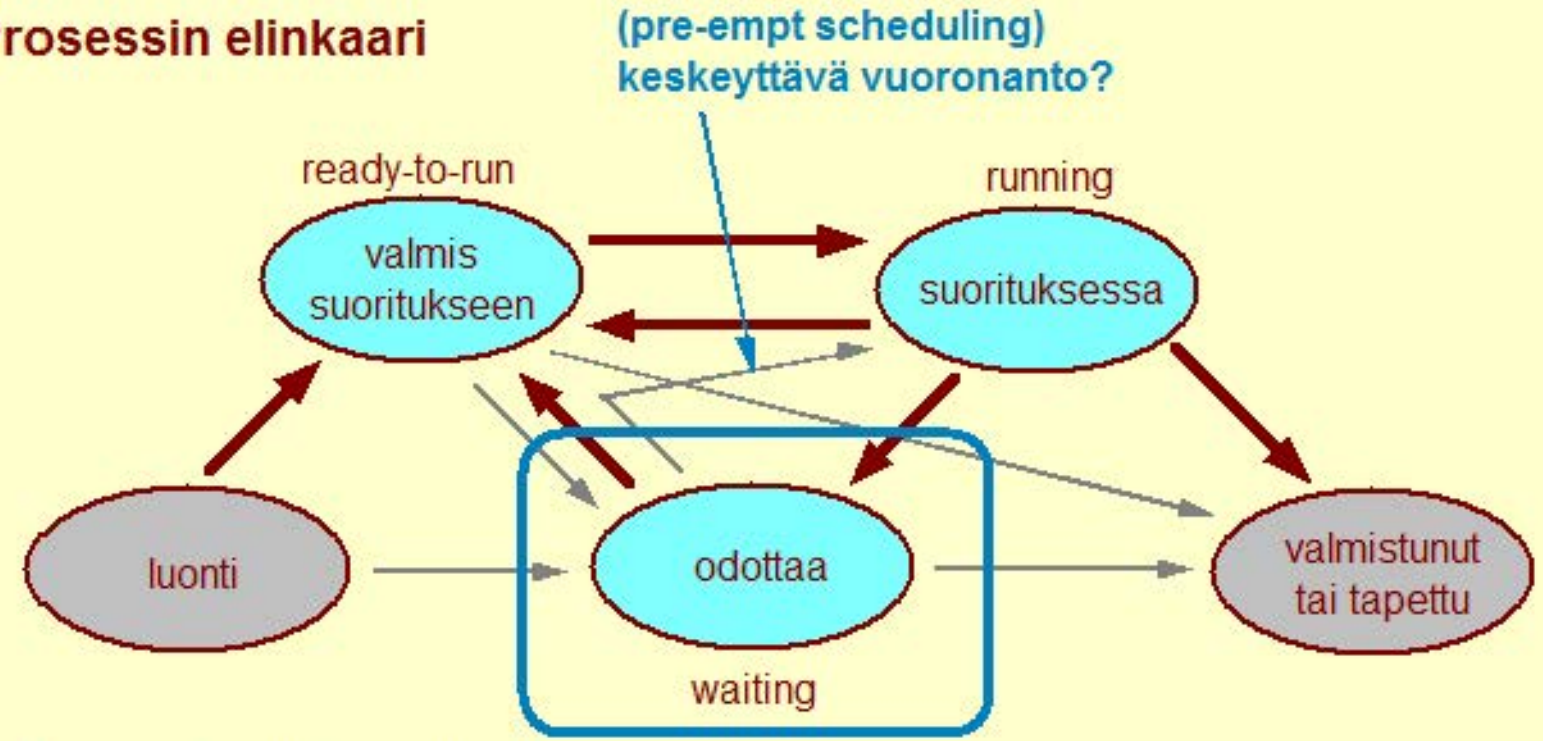
- mitä tila tarkoittaa?
- milloin tilanvaihto tapahtuu?
- mitä tilanvaihdossa tapahtuu?

Mitä suoritin tietää suorituksessa olevasta prosessista?

Copyright Teemu Kerola 2005

Suorituksessa on vain yksi prosessi kerrallaan. Suorittimen laiteympäristö eli sen kaikki rekisterit sisältävät nyt juuri ajossa olevan prosessin tietoja. Jos prosessin suoritus päättyy, niin kaikki prosessin käyttämät resurssit vapautetaan ja lopulta myös sen hallintotiedot poistetaan järjestelmästä. Useimmiten kuitenkin prosessi siirtyy odotustilaan odottamaan esimerkiksi jotain I/O-tapahtumaa. On myös mahdollista, että käyttöjärjestelmä herää henkiin ns. kellolaitekeskeytyksen avulla ja päättää, että suoritin pitää välillä antaa seuraavalle ready-to-run jonottajalle. Tällöin suorituksessa ollut prosessi viedään takaisin ready-to-run -jonoon odottamaan uutta vuoroa.

Prosessin elinkaari



Prosessin viisi suoritustilaa

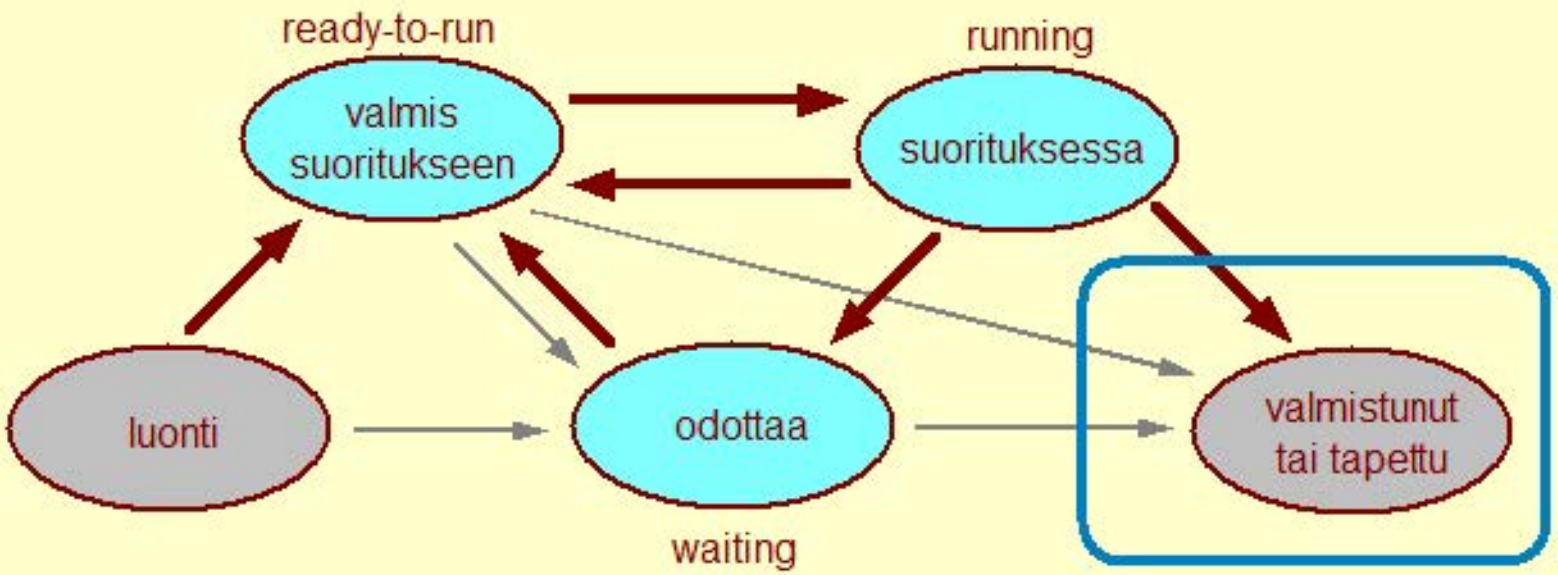
- mitä tila tarkoittaa?
- milloin tilanvaihto tapahtuu?
- mitä tilanvaihdossa tapahtuu?

Mitä suoritin tietää suorituksessa olevasta prosessista?

Copyright Teemu Kerola 2005

Odotustilassa prosessi odottaa yhtä tai useampaa tapahtumaa, jonka tapahduttua se olisi jälleen suorituskelpoinen. Käyttäjäprosesseille tyypillinen odotuksen syy on näppäimistöltä tulleen käyttäjän syötteen odottaminen. Tätäkin verkkoluentoa pyörittävä prosessi odottaa juuri nyt sinun seuraavaa näppäimistön tai hiiren painallustasi. Kun odotettava tapahtuma on tapahtunut, odottava prosessi siirretään ready-to-run -jonoon odottamaan suoritusvuoroa suorittimelle. Joissakin järjestelmissä on myös mahdollista, että juuri ready-to-run jonoon saapunut tärkeä prosessi kaappaakin suoritusvuoron suorittimella ja pääsee täten heti suoritukseen. Keskeytynyt prosessi siirretään tietenkin tällöin ready-to-run -jonoon odottamaan uutta suoritusvuoroa.

Prosessin elinkaari



Prosessin viisi suoritustilaa

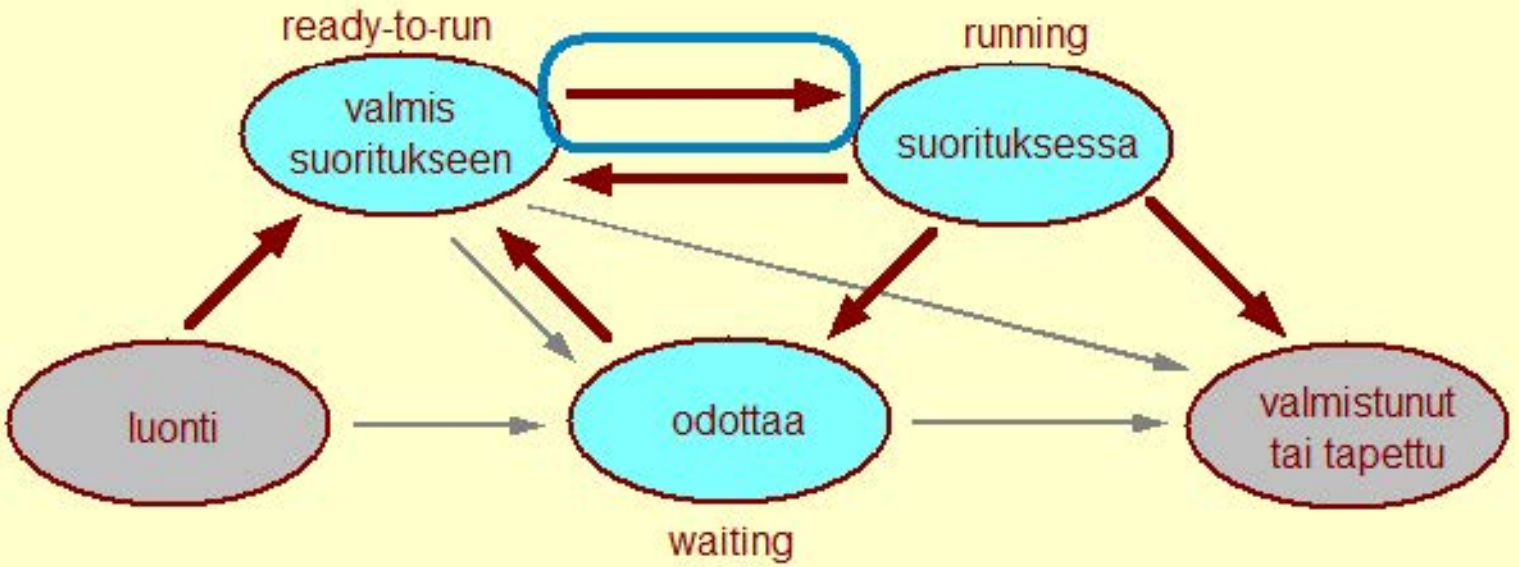
- mitä tila tarkoittaa?
- milloin tilanvaihto tapahtuu?
- mitä tilanvaihdossa tapahtuu?

Mitä suoritin tietää suorituksessa olevasta prosessista?

Copyright Teemu Kerola 2005

Päättyneen prosessin voidaan kuvitella olevan valmistunut-tilassa, vaikka todellisuudessa sitä ei ole enää järjestelmässä lainkaan. Heti kun kaikki prosessin resurssit on vapautettu (esim. aukiolevat tiedostot on suljettu ja muistialueet vapautettu uusiokäyttöön), myös prosessin käyttämät hallintotiedot nollataan ja vapautetaan uusiokäyttöön. Päättyneet prosessit eivät siis kuluta mitään resursseja. Prosessi päättyy normaalisti, jos sen pääohjelma suoritetaan loppuun. Epänormaali päättyminen taas tapahtuu, jos käyttöjärjestelmä (ehkä käyttäjän kehotteesta) tappaa ohjelman. Linux'in kill-komento on juuri tätä tarkoitusta varten.

Prosessin elinkaari



Prosessin viisi suoritustilaa

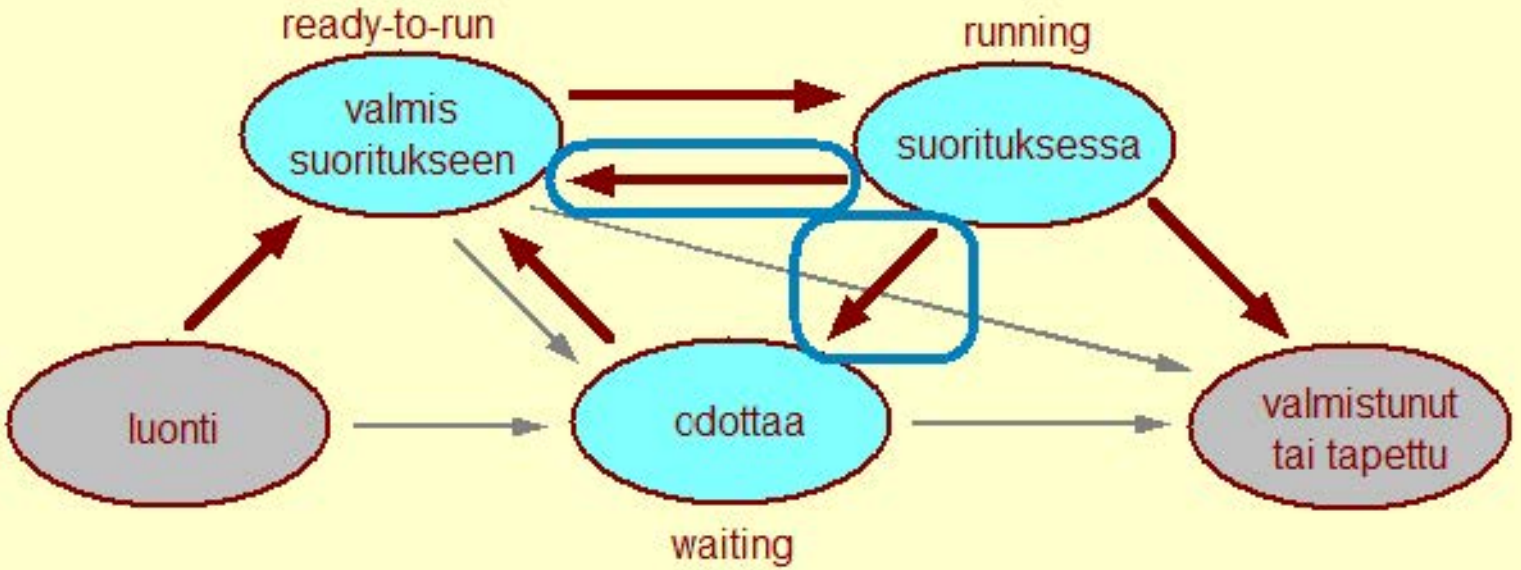
- mitä tila tarkoittaa?
- milloin tilanvaihto tapahtuu?
- mitä tilanvaihdossa tapahtuu?

Mitä suoritin tietää suorituksessa olevasta prosessista?

Copyright Teemu Kerola 2005

Vuoronanto tarkoittaa siis sitä tapahtumaa, kun käyttöjärjestelmän vuoronantaja valitsee jonkin ready-to-run prosessin ja siirtää sen suoritukseen suorittimelle. Suorittimen rekisterit ladataan kyseisen prosessin tiedoilla ja lopulta suoritus jatkuu juuri samasta konekäskystä, missä se aikaisemmin ehkä keskeytyi. Uudelle prosessille suoritus alkaa tietenkin pääohjelman ensimmäisestä käskystä. Keskeytyneen prosessin kaikkien olennaisten rekistereiden arvot ovat suorituksen jälleen jatkuessa täsmälleen samat kuin mitä ne olivat suorituksen keskeytyessä.

Prosessin elinkaari



Prosessin viisi suoritustilaa

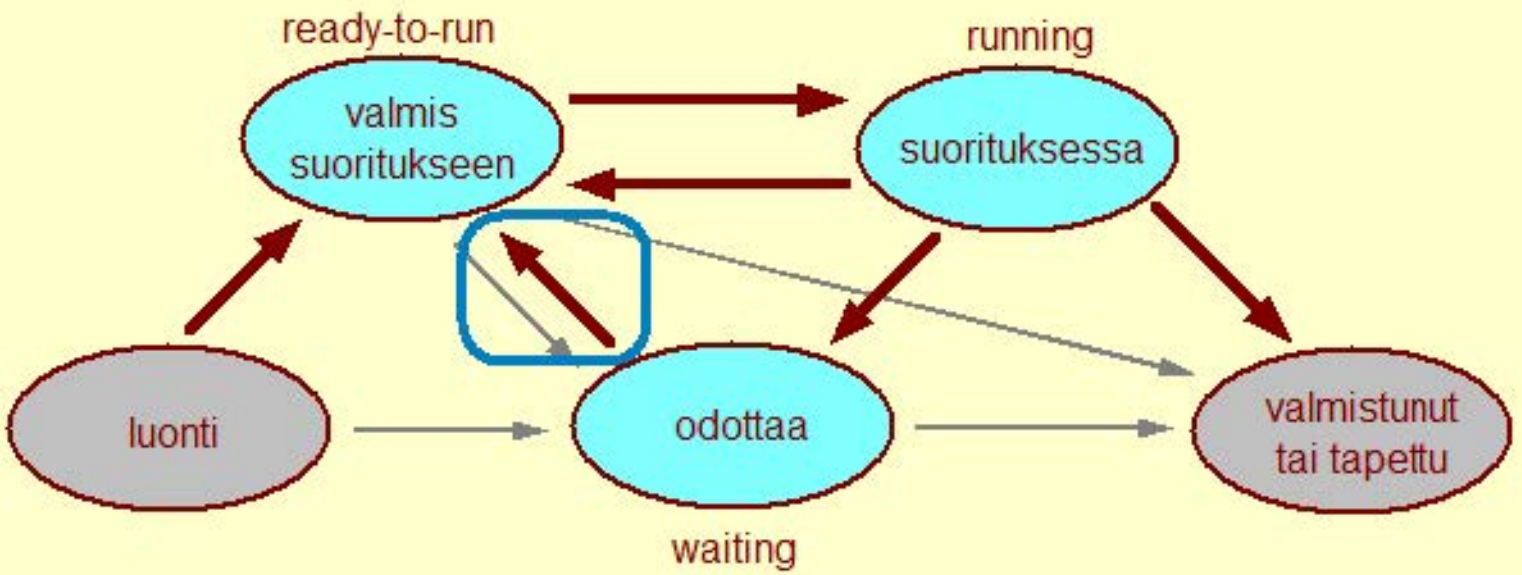
- mitä tila tarkoittaa?
- milloin tilanvaihto tapahtuu?
- mitä tilanvaihdossa tapahtuu?

Mitä suoritin tietää suorituksessa olevasta prosessista?

Copyright Teemu Kerola 2005

Kun prosessi menettää suoritusvuoron suorittimella, käyttöjärjestelmän vuoronantaja kopioi prosessin kaikki rekisterit eli suoritinympäristön talteen muistiin ja siirtää prosessin johonkin jonoon odottamaan joko jotain tapahtumaa tai sitten vain seuraavaa vuoroa suorittimelle. Käyttäjärjestelmä käsittelee kutakin prosessia yhtenä kokonaisuutena ja siirtelee sitä jonosta toiseen prosessin tilan mukaisesti. Suoritustilaa ei vastaa mikään varsinainen jono, vaikka loogisesti sellainen olisi helppo kuvitella. Mutta muita tiloja vastaavat jonot on käyttöjärjestelmässä toteutettu ihan tavallisina jonotietorakenteina.

Prosessin elinkaari



Prosessin viisi suoritustilaa

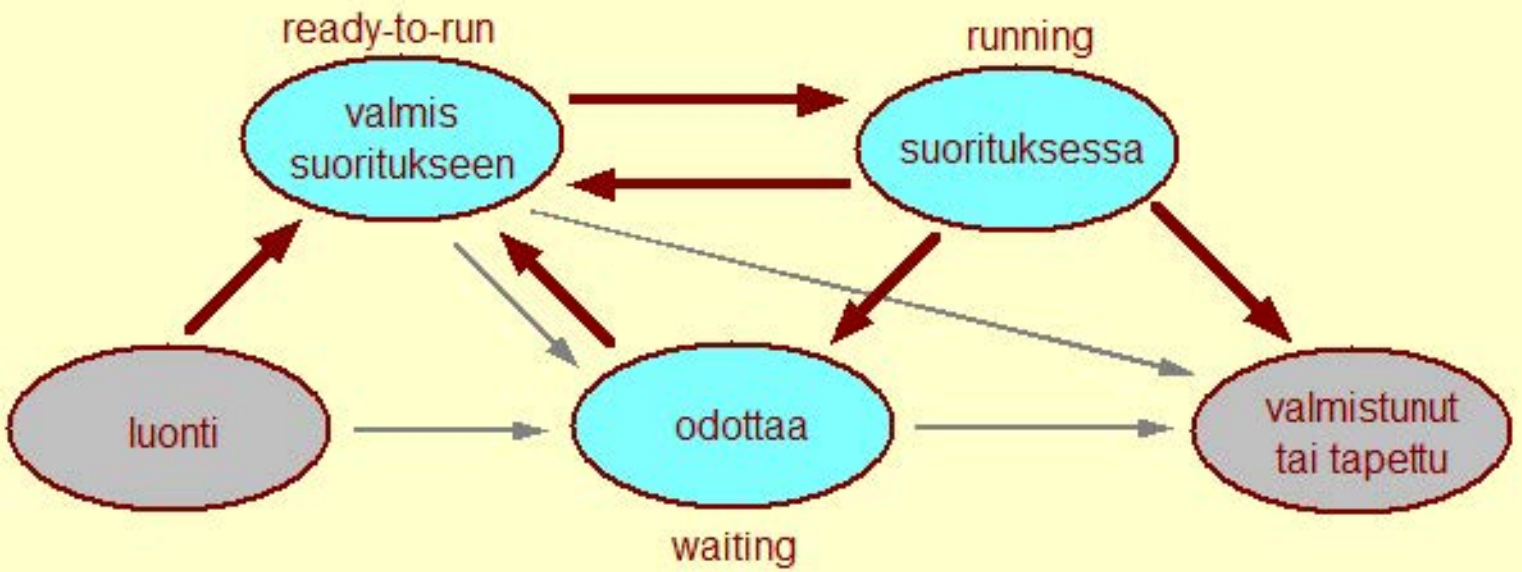
- mitä tila tarkoittaa?
- milloin tilanvaihto tapahtuu?
- mitä tilanvaihdossa tapahtuu?

Mitä suoritin tietää suorituksessa olevasta prosessista?

Copyright Teemu Kerola 2005

Kun prosessin odottama tapahtuma lopulta tapahtuu, niin käyttöjärjestelmä ilmoittaa siitä odottavalle prosessille yksinkertaisesti siirtämällä kyseinen prosessi ready-to-run -jonoon. Ilmoitus toteutetaan siis käytännössä vain siirtämällä kyseistä prosessia vastaava hallinto-olio eli prosessin kontrollilohko yhdestä jonosta toiseen. Esimerkiksi, jos nyt seuraavaksi painat vaikkapa TAB-näppäintä, niin tämän verkkoluennon toteuttava prosessi siirretään tässä järjestelmässä ready-to-run -jonoon, ja sitten vähän ajan päästä se myös saa suoritusvuoron ja näyttää sinulle seuraavan näkymän.

Prosessin elinkaari



Prosessin viisi suoritustilaa

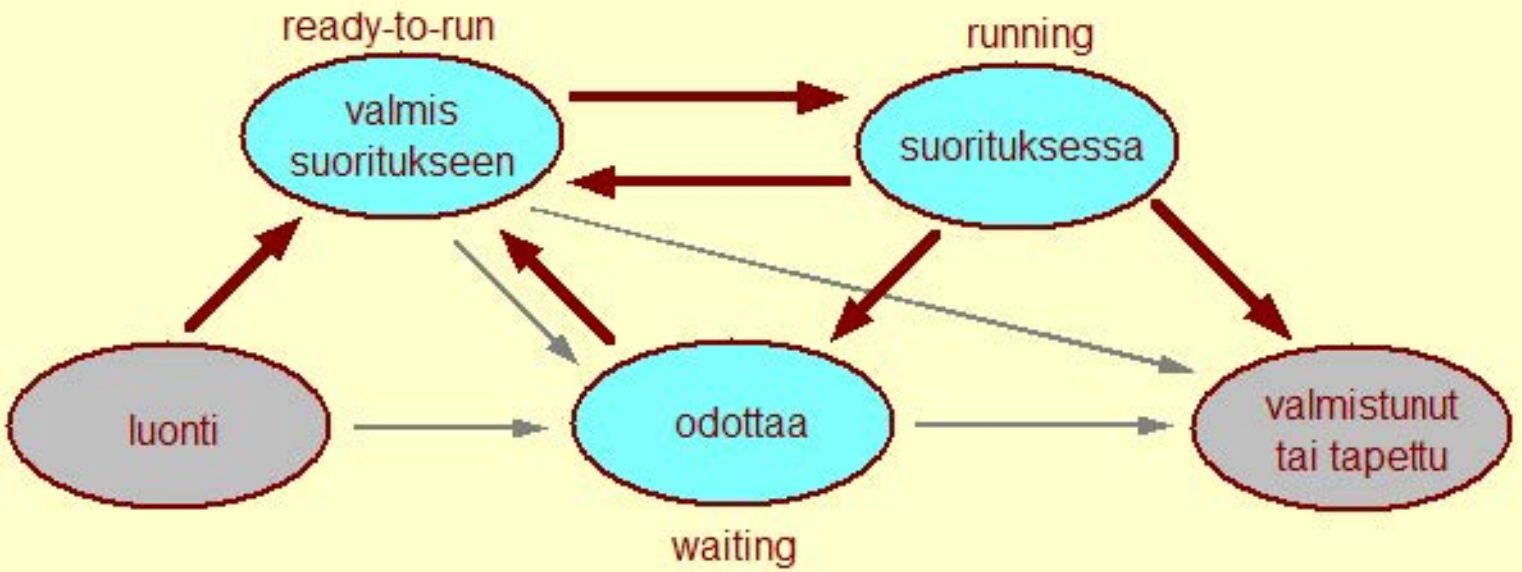
- mitä tila tarkoittaa?
- milloin tilanvaihto tapahtuu?
- mitä tilanvaihdossa tapahtuu?

Mitä suoritin tietää suorituksessa olevasta prosessista?

Copyright Teemu Kerola 2005

Suoritin suorittaa siis yhtä prosessia kerrallaan, mutta mitä suoritin tietää kyseisestä prosessista? Pohdi asiaa vähän aikaa, ja tarkista sitten vastauksesi seuraavasta näkymästä.

Prosessin elinkaari



Prosessin viisi suoritustilaa

- mitä tila tarkoittaa?
- milloin tilanvaihto tapahtuu?
- mitä tilanvaihdossa tapahtuu?

Mitä suoritin tietää suorituksessa olevasta prosessista?

Copyright Teemu Kerola 2005

Vastaus on ehkä vähän yllättäen, että 'ei mitään!' Suoritin ei itse asiassa tiedä mitään edes koko prosessin käsitteestä, joka on vain käyttöjärjestelmän luoma ja tuntema tietorakenne. Suoritin vai suorittaa konekäskyjä yksi kerrallaan eikä missään vaiheessa näe tai tunnista konekäskyjä suurempia kokonaisuuksia. Jos jossakin vaiheessa vuoronantaja lopulta sijoittaa vuoroon tulevan prosessin paikanlaskurin arvon PC-rekisterin arvoksi, niin vuoronantajan mielestä prosessinvaihto on saatu päätökseen. Suoritin taas vaan hakee seuraavan suoritettavan käskyn PC:n osoittamasta paikasta ja jatkaa käskyjen suorittamista niin kuin aina.

PCB - Prosessin esitysmuoto järjestelmässä

Prosessin kuvaaja eli kontrollilohko (Process Control Block, PCB)

Isohko tietue, joka sisältää kaiken yhdestä prosessista

- muistialueet, aukiolevat tiedostot, tiedostojen käsittelykohdat
- ei-suorituksessa oleville prosesseille myös suoritinympäristö
  - laiterekisterit, MMU:n rekisterit, kontrollirekisterit, PC

Joka prosessilla on oma PCB

Käyttöjärjestelmän prosessien hallinta rakentaa PCB:n prosessin luonnin yhteydessä ja tuhoaa sen prosessin päättyessä

- prosessi itse ei pääse käsiksi omaan PCB:nsä

Käyttöjärjestelmärutiinit käsittelevät prosessia sen PCB:n avulla

Copyright Teemu Kerola 2005

Prosessia edustaa järjestelmässä isohko tietorakenne, prosessin kuvaaja eli PCB. Rakenteessa sinällään ei ole mitään erikoista - se vain on suht'koht suuri, koska siinä on listattu kaikki mahdollinen tieto yhdestä prosessista. Idea on, että mitä tahansa tietoa prosessista halutaan säilyttää, se pidetään PCB:ssä. Siellä on yleistä hallintotietoa kaikista prosessin käyttämistä resursseista ja siellä on myös talletusalue prosessin suoritinympäristölle, jonne kaikki suoritinrekistereiden arvot talletetaan prosessin ollessa odotustilassa.

PCB - Prosessin esitysmuoto järjestelmässä

Prosessin kuvaaja eli kontrollilohko (Process Control Block, PCB)

Isohko tietue, joka sisältää kaiken yhdestä prosessista

- muistialueet, aukiolevat tiedostot, tiedostojen käsittelykohdat
- ei-suorituksessa oleville prosesseille myös suoritinympäristö
  - laiterekisterit, MMU:n rekisterit, kontrollirekisterit, PC

Joka prosessilla on oma PCB

Käyttöjärjestelmän prosessien hallinta rakentaa PCB:n prosessin luonnin yhteydessä ja tuhoaa sen prosessin päättyessä

- prosessi itse ei pääse käsiksi omaan PCB:nsä

Käyttöjärjestelmärutiinit käsittelevät prosessia sen PCB:n avulla

Copyright Teemu Kerola 2005

Joka prosessilla on siis oma PCB ja yleisesti ottaen se sijaitsee aina muistissa, jotta käyttöjärjestelmä voi sitä helposti ja nopeasti käsitellä. Jos prosessi joutuu odottamaan kauan aikaa, niin on myös mahdollista, että jotkut prosessin muistista varaamat alueet viedään levylle odottamaan, mutta PCB:n ydinosat pidetään kuitenkin aina muistissa.

PCB - Prosessin esitysmuoto järjestelmässä

Prosessin kuvaaja eli kontrollilohko (Process Control Block, PCB)

Isohko tietue, joka sisältää kaiken yhdestä prosessista

- muistialueet, aukiolevat tiedostot, tiedostojen käsittelykohdat
- ei-suorituksessa oleville prosesseille myös suoritinympäristö
  - laiterekisterit, MMU:n rekisterit, kontrollirekisterit, PC

Joka prosessilla on oma PCB

Käyttöjärjestelmän prosessien hallinta rakentaa PCB:n prosessin luonnin yhteydessä ja tuhoaa sen prosessin päättyessä

- prosessi itse ei pääse käsiksi omaan PCB:nsä

Käyttöjärjestelmärutiinit käsittelevät prosessia sen PCB:n avulla

Copyright Teemu Kerola 2005

Prosessin luominen tarkoittaa juuri PCB:n alustamista uuden prosessin tiedoilla. Käyttöjärjestelmän prosessienhallinta tekee tämän käyttäen tavallisia konekäskyjä. PCB:t talletetaan kuitenkin käyttöjärjestelmän etuoikeutetulle muistialueelle, jonne käyttäjätilassa suorittavat tavalliset ohjelmat eivät pääse käsiksi. Kun prosessi päättyy (tai se tapetaan), käyttöjärjestelmän prosessienhallinta vapauttaa lopuksi myös koko PCB-tietueen varaaman muistialueen uusiokäyttöä varten. Tässä vaiheessa juuri päättyneestä prosessista ei enää ole minkäänlaisia tietoja järjestelmässä, mahdollisia lokitiedostoja lukuunottamatta.

PCB - Prosessin esitysmuoto järjestelmässä

Prosessin kuvaaja eli kontrollilohko (Process Control Block, PCB)

Isohko tietue, joka sisältää kaiken yhdestä prosessista

- muistialueet, aukiolevat tiedostot, tiedostojen käsittelykohdat
- ei-suorituksessa oleville prosesseille myös suoritinympäristö
  - laiterekisterit, MMU:n rekisterit, kontrollirekisterit, PC

Joka prosessilla on oma PCB

Käyttöjärjestelmän prosessien hallinta rakentaa PCB:n prosessin luonnin yhteydessä ja tuhoaa sen prosessin päättyessä

- prosessi itse ei pääse käsiksi omaan PCB:nsä

Käyttöjärjestelmärutiinit käsittelevät prosessia sen PCB:n avulla

Copyright Teemu Kerola 2005

Kaikki prosessin käsittely tapahtuu siis PCB:n avulla. Käsittely tapahtuu joko muuttamalla PCB:ssä olevia tietoja tai siirtelemällä PCB:tä jonosta toiseen. Esimerkiksi, jos prosessi avaa uuden tiedoston lukemista varten, niin käyttöjärjestelmä ensin tarkistaa prosessin oikeuden tähän operaatioon ja sitten, jos kaikki oli kunnossa, suorittaa tiedoston avauksen ja merkitsee sen avatuksi tälle prosessille. Toisaalta, jos samalla kertaa tiedostoa pitää lukea, niin käyttöjärjestelmä siirtää tämän prosessin sopivaan jonoon odottamaan tiedoston luvun valmistumista.

Prosessin kuvaajan sisältö

- Prosessin tunniste — 14023
- Prosessin prioriteetti suorittimen vuoronantoa varten — 143
- Prosessin tila ja/tai odottamisen syy — odottaa, Disk 1, odottaa I/O:ta
- Prosessin suoritinympäristö, jos prosessi ei ole suorituksessa
  - suorittimen rekistereiden arvot (laskentarekisterit, muistinhallintarekisterit, tilarekisterit, SP, FP, PC, jne)

```
proc_env {
  R0, R1, R2, R3, R4, R5, SP, FP,
  BASE, LIMIT, SR, TR?
  PC
  }
```

- Poikkeuskäsittelijöiden osoitteet — "Java throw" = 43
  - ellei oletusarvoiset
- Aikaviipale — 15 ms
- Käytössä olevat muistialueet — code: {...}, data: {...}
- Aukiolevat tiedostot ja niiden käsittelykohdat — myfile, r, buffer5, readpointer = 136
- Käyttöjärjestelmän hallintotietoa — start 13:45:32 , cpu 23.45 s, in 65432 bytes
  - kokonaisaika, suoritinaika, verkkosiirännän määrä, jne
- Minix esimerkki *struct proc*

Copyright Teemu Kerola 2005

Prosessin kuvaajassa olevat tiedot vaihtelevat järjestelmästä toiseen, mutta lähes kaikista järjestelmistä löytyy ainakin seuraavat osat. Kukin prosessi tunnistetaan järjestelmässä yksikäsitteisellä 4- tai 5-numeroisella luvulla. Tämä luku on prosessin tunniste sen elinajan ja se voi toimia myös prosessin 'kahvana' (handle) käyttöjärjestelmärutiineille. Esimerkiksi Linux komentotulkille annettu komento 'kill 12345' pyytää komentotulkkia tappamaan prosessin 12345.

Prosessin kuvaajan sisältö

Prosessin tunniste — 14023

Prosessin prioriteetti suorittimen vuoronantoa varten — 143

Prosessin tila ja/tai odottamisen syy — odottaa, Disk 1, odottaa I/O:ta

Prosessin suoritinympäristö, jos prosessi ei ole suorituksessa

- suorittimen rekistereiden arvot (laskentarekisterit, muistinhallintarekisterit, tilarekisterit, SP, FP, PC, jne)

```
proc_env {
  R0, R1, R2, R3, R4, R5, SP, FP,
  BASE, LIMIT, SR, TR?
  PC
  }
```

Poikkeuskäsittelijöiden osoitteet — "Java throw" = 43

- ellei oletusarvoiset

Aikaviipale — 15 ms

Käytössä olevat muistialueet — code: {...}, data: {...}

Aukiolevat tiedostot ja niiden käsittelykohdat — myfile, r, buffer5, readpointer = 136

Käyttöjärjestelmän hallintotietoa — start 13:45:32 , cpu 23.45 s, in 65432 bytes

- kokonaisaika, suoritinaika, verkkosiirrännän määrä, jne

Minix esimerkki *struct proc*

Copyright Teemu Kerola 2005

Prosesseilla voi myös olla niiden tärkeyttä ilmaiseva prioriteetti, mikä kuitenkin vaikuttaa yleensä ainoastaan suorittimen vuoronannossa. Se ei siis vaikuta esimerkiksi I/O-laitteiden käyttöön.

Prosessin kuvaajan sisältö

Prosessin tunniste — 14023

Prosessin prioriteetti suorittimen vuoronantoa varten — 143

Prosessin tila ja/tai odottamisen syy — odottaa, Disk 1, odottaa I/O:ta

Prosessin suoritinympäristö, jos prosessi ei ole suorituksessa

- suorittimen rekistereiden arvot (laskentarekisterit, muistinhallintarekisterit, tilarekisterit, SP, FP, PC, jne)

```
proc_env {
  R0, R1, R2, R3, R4, R5, SP, FP,
  BASE, LIMIT, SR, TR?
  PC
  }
```

Poikkeuskäsittelijöiden osoitteet — "Java throw" = 43

- ellei oletusarvoiset

Aikaviipale — 15 ms

Käytössä olevat muistialueet — code: {...}, data: {...}

Aukiolevat tiedostot ja niiden käsittelykohdat — myfile, r, buffer5, readpointer = 136

Käyttöjärjestelmän hallintotietoa — start 13:45:32 , cpu 23.45 s, in 65432 bytes

- kokonaisaika, suoritinaika, verkkosiirrännän määrä, jne

Minix esimerkki *struct proc*

Copyright Teemu Kerola 2005

Odottaville prosesseille voi löytyä odottamisen syy selkeästi kirjattuna, tai sitten odottamisen syy ilmenee siitä, missä jonossa prosessi sattuu odottamaan.

Prosessin kuvaajan sisältö

Prosessin tunniste — 14023

Prosessin prioriteetti suorittimen vuoronantoa varten — 143

Prosessin tila ja/tai odottamisen syy — odottaa, Disk 1, odottaa I/O:ta

Prosessin suoritinympäristö, jos prosessi ei ole suorituksessa

- suorittimen rekistereiden arvot (laskentarekisterit, muistinhallintarekisterit, tilarekisterit, SP, FP, PC, jne)

```
proc_env {
  R0, R1, R2, R3, R4, R5, SP, FP,
  BASE, LIMIT, SR, TR?
  PC
  }
```

Poikkeuskäsittelijöiden osoitteet — "Java throw" = 43

- ellei oletusarvoiset

Aikaviipale — 15 ms

Käytössä olevat muistialueet — code: {...}, data: {...}

Aukiolevat tiedostot ja niiden käsittelykohdat — myfile, r, buffer5, readpointer = 136

Käyttöjärjestelmän hallintotietoa — start 13:45:32 , cpu 23.45 s, in 65432 bytes

- kokonaisaika, suoritinaika, verkkosiirännän määrä, jne

Minix esimerkki *struct proc*

Copyright Teemu Kerola 2005

Kuten aikaisemmin jo mainitsin, prosessin suoritinympäristö on odotusaikana jossain päin muistia tallessa. Jos suoritinympäristö on pieni, se voidaaan tallettaa suoraan kuvaajaan, mutta muussa tapauksessa kuvaajaan talletetaan vain suoritinympäristön osoite.

Prosessin kuvaajan sisältö

Prosessin tunniste — 14023

Prosessin prioriteetti suorittimen vuoronantoa varten — 143

Prosessin tila ja/tai odottamisen syy — odottaa, Disk 1, odottaa I/O:ta

Prosessin suoritinympäristö, jos prosessi ei ole suorituksessa

- suorittimen rekistereiden arvot (laskentarekisterit, muistinhallintarekisterit, tilarekisterit, SP, FP, PC, jne)

```
proc_env {
  R0, R1, R2, R3, R4, R5, SP, FP,
  BASE, LIMIT, SR, TR?
  PC
  }
```

Poikkeuskäsittelijöiden osoitteet — "Java throw" = 43

- ellei oletusarvoiset

Aikaviipale — 15 ms

Käytössä olevat muistialueet — code: {...}, data: {...}

Aukiolevat tiedostot ja niiden käsittelykohdat — myfile, r, buffer5, readpointer = 136

Käyttöjärjestelmän hallintotietoa — start 13:45:32 , cpu 23.45 s, in 65432 bytes

- kokonaisaika, suoritinaika, verkkosiirrännän määrä, jne

Minix esimerkki *struct proc*

Copyright Teemu Kerola 2005

Poikkeustilanteiden käsittelyä varten kaikkien mahdollisten poikkeuskäsittelijöiden osoitteet täytyy olla tietenkin tiedossa. Joissakin järjestelmissä voi prosessin alustuksen yhteydessä vaihtaa jotkut oletusarvoiset poikkeuskäsittelijät tämän prosessin omiin käsittelijöihin, ja tällainen tieto talletetaan tietenkin myös PCB:hen. Esimerkiksi Java-ohjelmissa on tyypillistä, että throw-catch ja try-catch -rakenteet toteutetaan Java-ympäristön omilla poikkeuskäsittelijöillä.

Prosessin kuvaajan sisältö

Prosessin tunniste — 14023

Prosessin prioriteetti suorittimen vuoronantoa varten — 143

Prosessin tila ja/tai odottamisen syy — odottaa, Disk 1, odottaa I/O:ta

Prosessin suoritinympäristö, jos prosessi ei ole suorituksessa

- suorittimen rekistereiden arvot (laskentarekisterit, muistinhallintarekisterit, tilarekisterit, SP, FP, PC, jne)

```
proc_env {
  R0, R1, R2, R3, R4, R5, SP, FP,
  BASE, LIMIT, SR, TR?
  PC
  }
```

Poikkeuskäsittelijöiden osoitteet — "Java throw" = 43

- ellei oletusarvoiset

Aikaviipale — 15 ms

Käytössä olevat muistialueet — code: {...}, data: {...}

Aukiolevat tiedostot ja niiden käsittelykohdat — myfile, r, buffer5, readpointer = 136

Käyttöjärjestelmän hallintotietoa — start 13:45:32 , cpu 23.45 s, in 65432 bytes

- kokonaisaika, suoritinaika, verkkosiirrännän määrä, jne

Minix esimerkki *struct proc*

Copyright Teemu Kerola 2005

Aikaviipaleella tarkoitetaan sitä aikaa, minkä prosessi voi korkeintaan pitää suoritinta hallussaan, ennen kuin se pitää välillä antaa muille Ready-to-run -jonossa odottaville prosesseille. Aikaviipale voi olla vakio tai sitten prosessikohtainen. On myös käyttöjärjestelmiä, joissa aikaviipaleita ei ole lainkaan ja prosessi voi pitää suorittimen hallussaan kunnes se siitä itse luopuu. Esim. Windowsin edeltäjä DOS-käyttöjärjestelmä oli tällainen.

Prosessin kuvaajan sisältö

Prosessin tunniste — 14023

Prosessin prioriteetti suorittimen vuoronantoa varten — 143

Prosessin tila ja/tai odottamisen syy — odottaa, Disk 1, odottaa I/O:ta

Prosessin suoritinympäristö, jos prosessi ei ole suorituksessa

- suorittimen rekistereiden arvot (laskentarekisterit, muistinhallintarekisterit, tilarekisterit, SP, FP, PC, jne)

```
proc_env {
  R0, R1, R2, R3, R4, R5, SP, FP,
  BASE, LIMIT, SR, TR?
  PC
  }
```

Poikkeuskäsittelijöiden osoitteet — "Java throw" = 43

- ellei oletusarvoiset

Aikaviipale — 15 ms

Käytössä olevat muistialueet — code: {...}, data: {...}

Aukiolevat tiedostot ja niiden käsittelykohdat — myfile, r, buffer5, readpointer = 136

Käyttöjärjestelmän hallintotietoa — start 13:45:32 , cpu 23.45 s, in 65432 bytes

- kokonaisaika, suoritinaika, verkkosiirännän määrä, jne

Minix esimerkki *struct proc*

Copyright Teemu Kerola 2005

Prosessin käyttämät muistialueet ovat myös tallessa PCB:ssä. Niiden avulla on helppo valvoa prosessin käyttämän muistitilan määrää ja prosessin päättyessä ne on helppo vapauttaa.

Prosessin kuvaajan sisältö

Prosessin tunniste — 14023

Prosessin prioriteetti suorittimen vuoronantoa varten — 143

Prosessin tila ja/tai odottamisen syy — odottaa, Disk 1, odottaa I/O:ta

Prosessin suoritinympäristö, jos prosessi ei ole suorituksessa

- suorittimen rekistereiden arvot (laskentarekisterit, muistinhallintarekisterit, tilarekisterit, SP, FP, PC, jne)

```
proc_env {
  R0, R1, R2, R3, R4, R5, SP, FP,
  BASE, LIMIT, SR, TR?
  PC
  }
```

Poikkeuskäsittelijöiden osoitteet — "Java throw" = 43

- ellei oletusarvoiset

Aikaviipale — 15 ms

Käytössä olevat muistialueet — code: {...}, data: {...}

Aukiolevat tiedostot ja niiden käsittelykohdat — myfile, r, buffer5, readpointer = 136

Käyttöjärjestelmän hallintotietoa — start 13:45:32 , cpu 23.45 s, in 65432 bytes

- kokonaisaika, suoritinaika, verkkosiirrännän määrä, jne

Minix esimerkki *struct proc*

Copyright Teemu Kerola 2005

Toinen tyypillinen prosessin käyttämä resurssi on sen käyttämät tiedostot. Kaikki aukiolevat tiedostot luetellaan ja jokaisesta kirjataan ainakin sen käyttötapa, puskurialueet ja käsittelykohta.

Prosessin kuvaajan sisältö

Prosessin tunniste — 14023

Prosessin prioriteetti suorittimen vuoronantoa varten — 143

Prosessin tila ja/tai odottamisen syy — odottaa, Disk 1, odottaa I/O:ta

Prosessin suoritinympäristö, jos prosessi ei ole suorituksessa

- suorittimen rekistereiden arvot (laskentarekisterit, muistinhallintarekisterit, tilarekisterit, SP, FP, PC, jne)

```
proc_env {
  R0, R1, R2, R3, R4, R5, SP, FP,
  BASE, LIMIT, SR, TR?
  PC
  }
```

Poikkeuskäsittelijöiden osoitteet — "Java throw" = 43

- ellei oletusarvoiset

Aikaviipale — 15 ms

Käytössä olevat muistialueet — code: {...}, data: {...}

Aukiolevat tiedostot ja niiden käsittelykohdat — myfile, r, buffer5, readpointer = 136

Käyttöjärjestelmän hallintotietoa — start 13:45:32 , cpu 23.45 s, in 65432 bytes

- kokonaisaika, suoritinaika, verkkosiirrännän määrä, jne

Minix esimerkki *struct proc*

Copyright Teemu Kerola 2005

PCB:ssä on myös yleistä järjestelmän hallintotietoa, mitä ei varsinaisesti tarvita prosessin suorittamiseen, mutta jota voidaan käyttää koko järjestelmän hallintaan tai laskutukseen. Nämä tiedot vaihtelevat suuresti järjestelmästä toiseen.

```
09410 struct proc {
09411   struct stackframe_s p_reg;  /* process' registers saved in stack frame */
09412
09414   reg_t p_ldt_sel;            /* selector in gdt giving ldt base and limit*/
09415   struct segdesc_s p_ldt[2];  /* local descriptors for code and data */
09416                               /* 2 is LDT_SIZE - avoid include protect.h */
09436   reg_t *p_stguard;           /* stack guard word */
09437
09438   int p_nr;                   /* number of this process (for fast access) */
09439
09440   int p_int_blocked;          /* nonzero if int msg blocked by busy task */
09441   int p_int_held;             /* nonzero if int msg held by busy syscall */
09442   struct proc *p_nextheld;    /* next in chain of held-up int processes */
09443
09444   int p_flags;                  /* P_SLOT_FREE, SENDING, RECEIVING, etc. */
09445   struct mem_map p_map[NR_SEGS];/* memory map */
09446   pid_t p_pid;                  /* process id passed in from MM */
09447
09448   clock_t user_time;            /* user time in ticks */
09449   clock_t sys_time;             /* sys time in ticks */
09450   clock_t child_utime;          /* cumulative user time of children */
09451   clock_t child_stime;          /* cumulative sys time of children */
09452   clock_t p_alarm;              /* time of next alarm in ticks, or 0 */
09453
```

Minix esimerkki *struct proc*

http://www.cs.vu.nl/~ast/minix.html (1.6.2005)
Copyright (c) 1987,1997, Prentice Hall, All rights reserved.
Copyright Teemu Kerola 2005

Andrew Tanenbaum on kehittänyt jo 1987 opetuskäyttöön sopivan UNIX-pohjaisen käyttöjärjestelmän Minix. Minix ei ole Linux'in kaltainen todellinen käyttöjärjes:elmä, vaan se on nimenomaan suunniteltu opetusnäkökulmasta. Järjestelmää ja sen lähdekoodia saa vapaasti käyttää opetustarkoituksiin. Ohessa on Minixin PCB Intelin suorittimelle. Tarkastelkaa PCB:tä (struct proc) yleisellä tasolla, älkääkä turhaan yrittäkö ymmärtää kaikkia detaljeja. Sieltä löytyy helposti prosessin tunnus (PID), muistialueet (mem_map), koodi- ja data-alueet (p_ldt) ja linkkikenttä (p_nextready) PCE:n sijoittamiseksi jonoon. Esittelen jatkossa myös muita Minix-esimerkkejä.

Prosessin tilanvaihto

Prosessin elinkaarimallin mukaisten tilojen vaihto tapahtuu siirtämällä prosessi (sen PCB) jonosta toiseen

ready-to-run -jono odottaa suoritinta

- yksi jono
- monta jonoa, yksi kullekin prioriteettiluokalle

running-jono

- ei oikeastaan ole olemassa kuin ajatuksena

waiting-jono

- joka odotustyypille oma jononsa
  - laitteen Disk1 I/O:n valmistumista odottavat
  - näppäimistön painallusta odottavat
  - kellolaitekeskeytystä odottavat
  - prosessilta 1345 signaalia odottavat

Minix esimerkki *ready*

http://www.cs.vu.nl/~ast/minix.html (1.6.2005)
Copyright (c) 1987,1997, Prentice Hall, All rights reserved.

Copyright Teemu Kerola 2005

Käyttöjärjestelmä siis hallinnoi prosesseja muokkaamalla niiden PCB:n sisältöä ja toteuttamalla prosessin tilanvaihdot siirtämällä prosesseja jonoista toiseen. Ready-to-run -jonoja on useissa järjestelmissä useita, yksi kutakin suorittimen prioriteettiluokkaa kohden. Vuoronantaja valitsee aina suoritukseen ensimmäisen prosessin korkeimman prioriteetin omaavasta ei-tyhjästä jonosta. Kun prosessi tulee valmiiksi suoritukseen, se sijoitetaan oman prioriteettiluokkansa jonon hännille.

Prosessin tilanvaihto

Prosessin elinkaarimallin mukaisten tilojen vaihto tapahtuu siirtämällä prosessi (sen PCB) jonosta toiseen

ready-to-run -jono
- yksi jono
- monta jonoa, yksi kullekin prioriteettiluokalle

running-jono — suoritustila
- ei oikeastaan ole olemassa kuin ajatuksena

waiting-jono
- joka odotustyypille oma jononsa
  - laitteen Disk1 I/O:n valmistumista odottavat
  - näppäimistön painallusta odottavat
  - kellolaitekeskeytystä odottavat
  - prosessilta 1345 signaalia odottavat

Minix esimerkki *ready*

http://www.cs.vu.nl/~ast/minix.html (1.6.2005)
Copyright (c) 1987,1997, Prentice Hall, All rights reserved.

Copyright Teemu Kerola 2005

Suorituksessa olevat prosessit eivät oikeastaan ole missään jonossa, mutta voimme helposti mieltää, että suorittimella on yhden prosessin jcno ja että kyseinen prosessi on nyt juuri sitten suorituksessa. Toisaalta, jos suorituksessa olevat prosessit ova: explisiittisesti jossakin jonossa, niin täytyy kuitenkin muistaa, että niiden PCB:ssä oleva suoritinympäristö ei ole ajan tasalla, koska se vastaa tilarnetta suoritusvuoron alkaessa. Tällä voi olla merkitystä nykyisillä moniprosessorikoneilla, joissa yhdellä suorittimella suorittava käyttöjärjestelmäprosessi voi tarkkailla samaan aikaan toisel a suorittimella suorituksessa olevan prosessin PCB:tä.

Prosessin tilanvaihto

Prosessin elinkaarimallin mukaisten tilojen vaihto tapahtuu siirtämällä prosessi (sen PCB) jonosta toiseen

ready-to-run -jono

- yksi jono
- monta jonoa, yksi kullekin prioriteettiluokalle

running-jono

- ei oikeastaan ole olemassa kuin ajatuksena

waiting-jono odotustila

- joka odotustyypille oma jononsa
  - laitteen Disk1 I/O:n valmistumista odottavat
  - näppäimistön painallusta odottavat
  - kellolaitekeskeytystä odottavat
  - prosessilta 1345 signaalia odottavat

Minix esimerkki *ready*

http://www.cs.vu.nl/~ast/minix.html (1.6.2005)
Copyright (c) 1987,1997, Prentice Hall, All rights reserved.

Copyright Teemu Kerola 2005

Odotustilaakaan ei vastaa vain yksi jono vaan useita. Jokaista erilaista odotuksen tyyppiä vastaa oma jononsa, josta juuri sitä tapahtumaa odottavat prosessit on sitten helppo löytää.

Prosessin tilanvaihto

```
24932 PRIVATE void ready(rp)
24933 register struct proc *rp;       /* this process is now runnable */
24934 {
24935 /* Add 'rp' to the end of one of the queues of runnable processes. Three
24936  * queues are maintained:
24937  *   TASK_Q   - (highest priority) for runnable tasks
24938  *   SERVER_Q - (middle priority) for MM and FS only
24939  *   USER_Q   - (lowest priority) for user processes
24940  */
24941
24942   if (istaskp(rp)) {
24943         if (rdy_head[TASK_Q] != NIL_PROC)
24944                 /* Add to tail of nonempty queue. */
24945                 rdy_tail[TASK_Q]->p_nextready = rp;
24946         else {
24947                 proc_ptr =              /* run fresh task next */
24948                 rdy_head[TASK_Q] = rp;  /* add to empty queue */
24949         }
24950         rdy_tail[TASK_Q] = rp;
24951         rp->p_nextready = NIL_PROC;     /* new entry has no successor */
24952         return;
24953   }
```

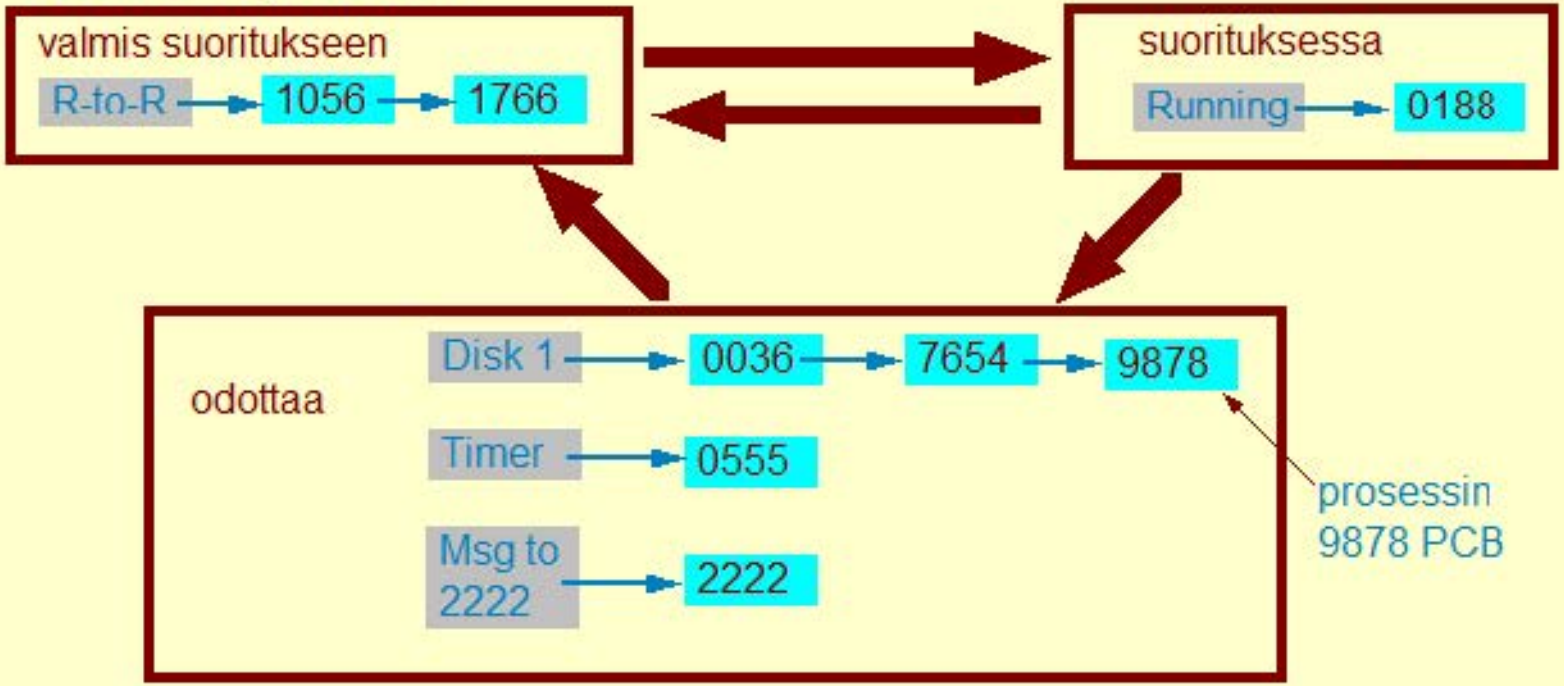
Minix esimerkki *ready*

http://www.cs.vu.nl/~ast/minix.html (1.6.2005)
Copyright (c) 1987,1997, Prentice Hall, All rights reserved.

Copyright Teemu Kerola 2005

Minix-esimerkin rutiini ready siirtää parametrina annetun prosessin kuvaajan rp johonkin kolmen eri prioriteettitason ready-to-run jonoon. Käyttöjärjestelmän taskeillä on suurin prioriteetti ja käyttäjätason prosesseilla pienin. Käyttöjärjestelmän muistinhallinta- ja tiedostojenhallintaprosesseilla on sitten vielä edellämainittujen väliin jäävä prioriteetti. Ready-to-run -jonojen käsittelyä nopeuttaa valmiit osoittimet sekä jonojen alkuun (rdy_head) että niiden loppuun (rdy_tail). PCB:n p_nextready-kenttää käytetään prosessien kuvaajien linkittämiseen toisiinsa.

Prosessit jonoissa



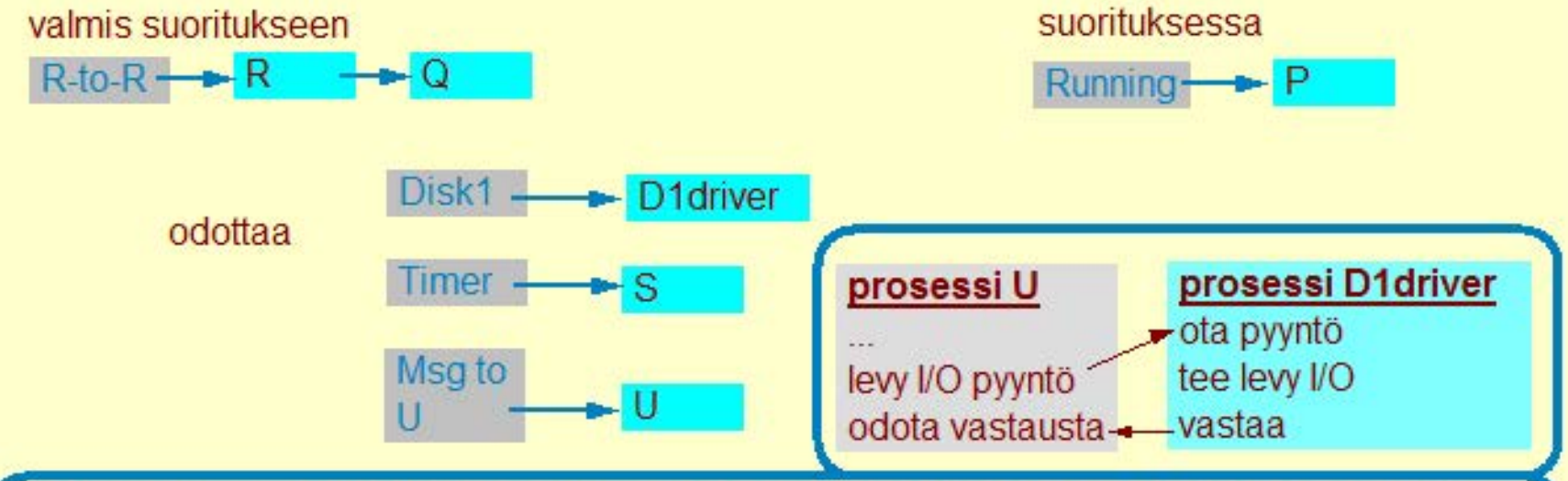
Vuoronanto: valitse seuraava prosessi ready-to-run jonosta (R-to-R) ja siirrä se suoritukseen CPU:lle

- kopioi ensin vanhan prosessin 0188 suoritinympäristö muistiin ja
- kopioi sitten uuden prosessin 1056 suoritinympäristö muistista rekistereihin

Copyright Teemu Kerola 2005

Käyttöjärjestelmän näkökulmasta prosessien hallinta on siis paljolti jonojen hallintaa. Kaikki prosessit (siis niiden PCB:t) ovat jossakin jonossa koko ajan, ja kukin jono indikoi suoraan siinä jonossa olevien prosessien tilan. Kuvan esimerkissä järjestelmässä on siis 7 prosessia, joista 2 odottaa tällä hetkellä suoritusvuoroa. Käyttöjärjestelmäprosessi 0555 odottaa kellolaitekeskeytystä, minkä jälkeen se pääsee pian suoritukseen tekemään omaa hallintahommaansa. Kolme prosessia on odottamassa levy I/O:n päättymistä, joten ilmeisesti järjestelmän levy on aika lailla kuormitettu. Prosessi 2222 odottaa, että joku antaisi sille luvan jatkaa työtään.

Esimerkki: I/O-laitekeskeytys



Lähtötilanne: U pyytänyt D1driveria lukemaan levyltä, D1driver odottaa I/O:ta

Disk1 antaa I/O-laitekeskeytyksen

- D1driver ready-to-run jonoon isolla prioriteetilla

D1driver pääsee heti suoritukseen

- ensin P takaisin jonoon ja sitten .... D1driver pääsee suoritukseen

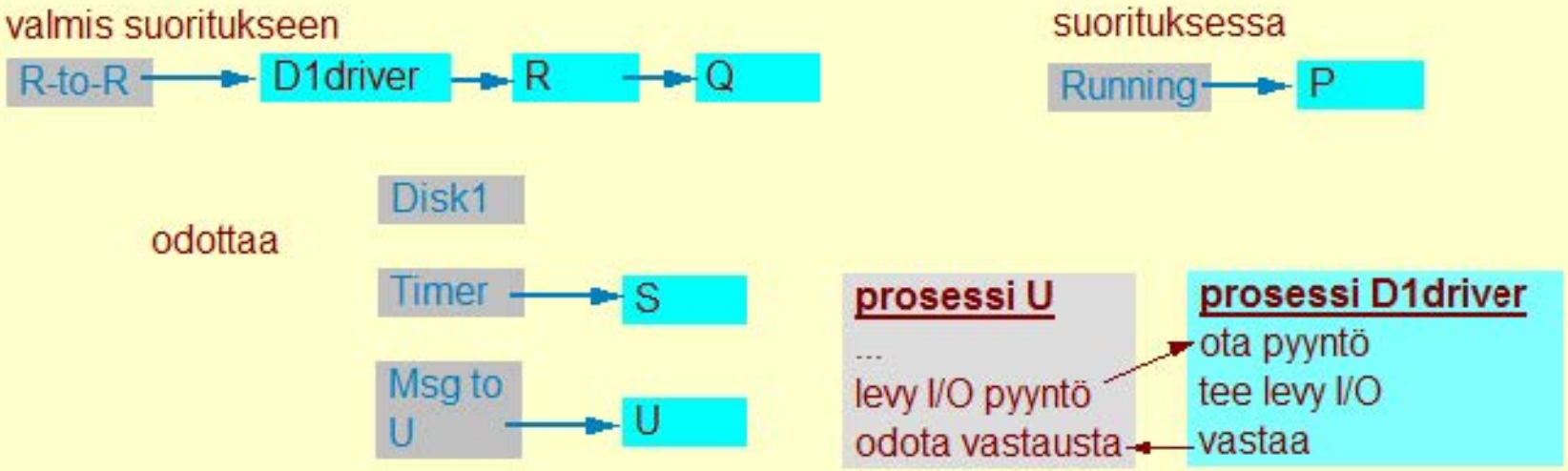
D1driver saa työn valmiiksi, U pääsee suoritusjonoon

- D1driver jää odottamaan uutta työtä, ja R pääsee suoritukseen

Copyright Teemu Kerola 2005

Tässä esimerkissä käyttäjätason prosessi U on pyytänyt käyttöjärjestelmään kuuluvaa levyn Disk1 laiteajuria D1driver suorittamaan levyltä lukua. U odottaa viestiä ajurilta työn valmiiksi saamisesta. D1driver on pyytänyt levykköä lukemaan datan muistiin ja odottaa luvun valmistumista levyn I/O-laitekeskeytysjonossa. Suoritusvuorossa on jokin muu prosessi P, jolla ei ole mitään tekemistä laiteajurin tai prosessin U kanssa. Käyttäjätason prosessit R ja Q odottavat suoritusvuoroa ready-to-run -jonossa.

Esimerkki: I/O-laitekeskeytys



Lähtötilanne: U pyytänyt D1driveria lukemaan levyltä, D1driver odottaa I/O:ta

Disk1 antaa I/O-laitekeskeytyksen

- D1driver ready-to-run jonoon isolla prioriteetilla

D1driver pääsee heti suoritukseen

- ensin P takaisin jonoon ja sitten .... D1driver pääsee suoritukseen

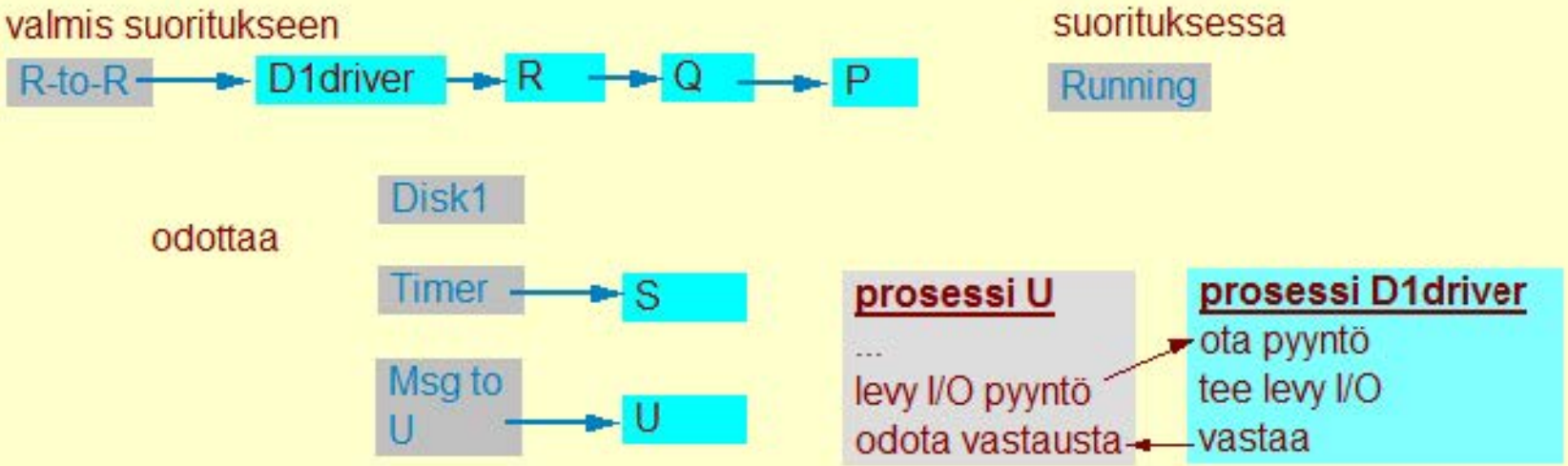
D1driver saa työn valmiiksi, U pääsee suoritusjonoon

- D1driver jää odottamaan uutta työtä, ja R pääsee suoritukseen

Copyright Teemu Kerola 2005

Laitekeskeytyksen tapahtuessa prosessin P suoritus keskeytyy ja kyseisen keskeytyksen keskeytyskäsittelijä pääsee suoritukseen P:n ympäristössä. Keskeytyskäsittelijä tutkii tilanteen ja siirtää laiteajurin D1driver ready-to-run jonoon oman prioriteettinsa mukaiselle paikalle. Käyttöjärjestelmäprosessina laiteajurilla on suurempi prioriteetti kuin käyttäjätason prosesseilla R tai Q. Jos käyttöjärjestelmä ei ole keskeyttävä tai P:n prioriteetti on suurempi kuin D1driver'in, niin P voi jatkaa suoritusta. Keskeyttävässä käyttöjärjestelmässä on myös mahdollista, että P menettää suoritusvuoron ja D1driver pääsee heti suoritukseen.

Esimerkki: I/O-laitekeskeytys



Lähtötilanne: U pyytänyt D1driveria lukemaan levyltä, D1driver odottaa I/O:ta

Disk1 antaa I/O-laitekeskeytyksen

- D1driver ready-to-run jonoon isolla prioriteetilla

D1driver pääsee heti suoritukseen

- ensin P takaisin jonoon ja sitten .... D1driver pääsee suoritukseen

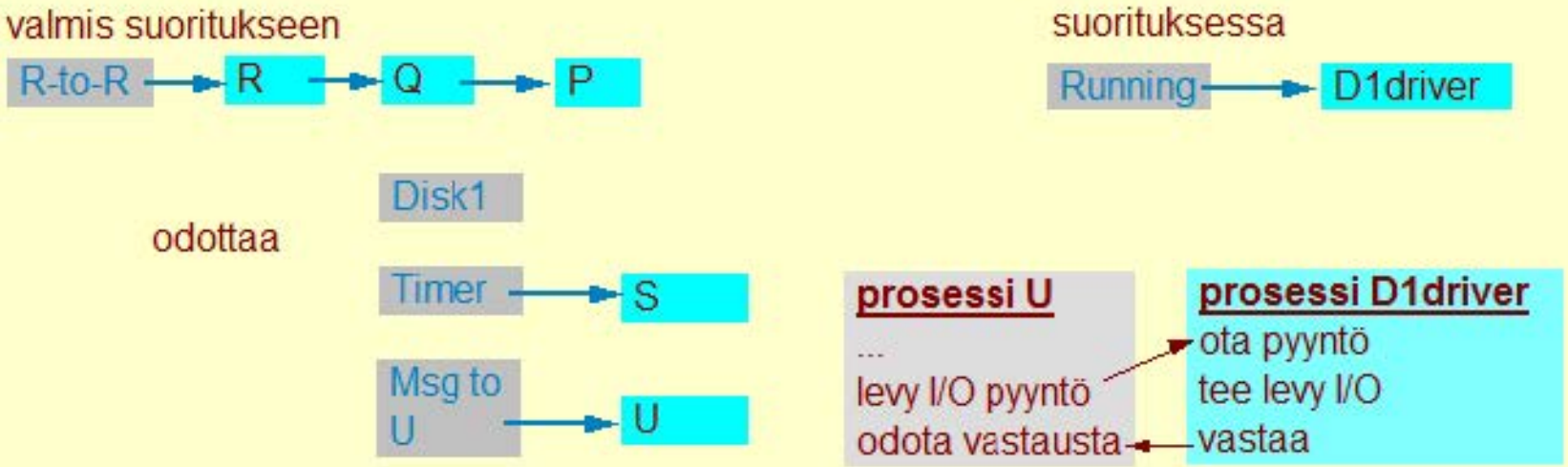
D1driver saa työn valmiiksi, U pääsee suoritusjonoon

- D1driver jää odottamaan uutta työtä, ja R pääsee suoritukseen

Copyright Teemu Kerola 2005

Oletetaan nyt, että käyttöjärjestelmä on keskeyttävä, eli suoritusvuoro voi milloin tahansa vaihtua korkeamman suoritinprioriteetin omaavan prosessin hyväksi. D1drver tuli juuri suorituskelpoiseksi ja sillä on suurempi prioriteetti kuin ennen keskeytystä suorituksessa olleella prosessilla P. D1driver pääsee siis heti suoritukseen. P siirretään ensin takaisin ready-to-run jonoon ja samalla sen suoritinympäristö kopioidaan P:n kuvaajaan talteen.

Esimerkki: I/O-laitekeskeytys



Lähtötilanne: U pyytänyt D1driveria lukemaan levyltä, D1driver odottaa I/O:ta

Disk1 antaa I/O-laitekeskeytyksen

- D1driver ready-to-run jonoon isolla prioriteetilla

D1driver pääsee heti suoritukseen

- ensin P takaisin jonoon ja sitten .... D1driver pääsee suoritukseen

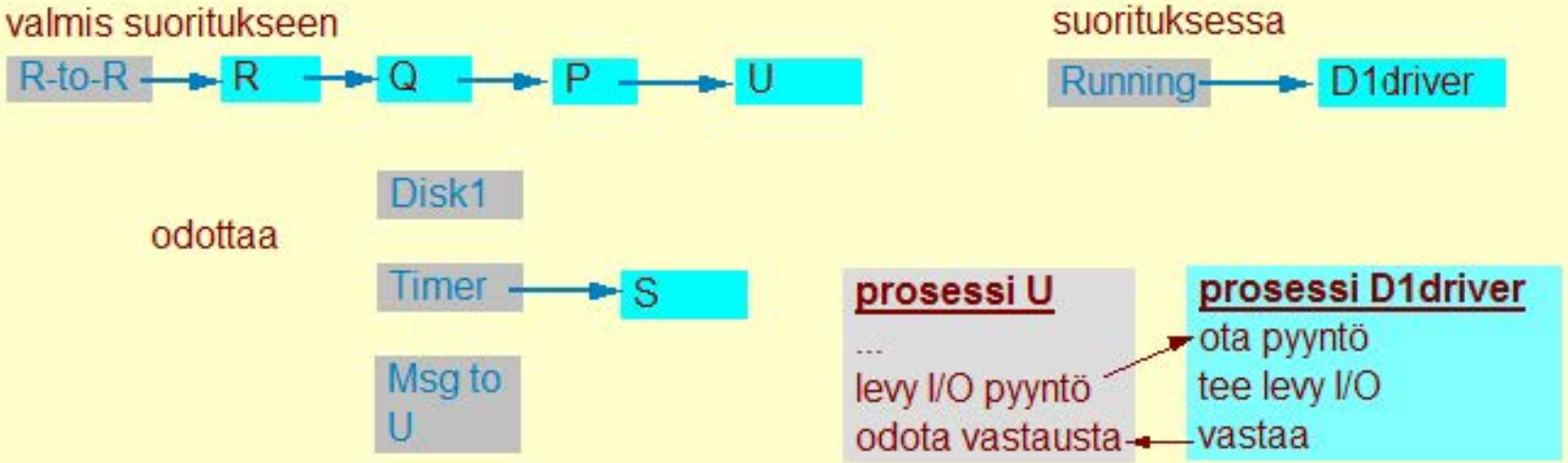
D1driver saa työn valmiiksi, U pääsee suoritusjonoon

- D1driver jää odottamaan uutta työtä, ja R pääsee suoritukseen

Copyright Teemu Kerola 2005

D1driver'in suoritinympäristö kopioidaan nyt sen PCB:stä suorittimelle ja sen suoritus alkaa. Laiteajuri rupeaa nyt tutkimaan, mitä tuo I/O-laitekeskeytys oikein tarkoitti.

Esimerkki: I/O-laitekeskeytys



Lähtötilanne: U pyytänyt D1driveria lukemaan levyltä, D1driver odottaa I/O:ta

Disk1 antaa I/O-laitekeskeytyksen

- D1driver ready-to-run jonoon isolla prioriteetilla

D1driver pääsee heti suoritukseen

- ensin P takaisin jonoon ja sitten .... D1driver pääsee suoritukseen

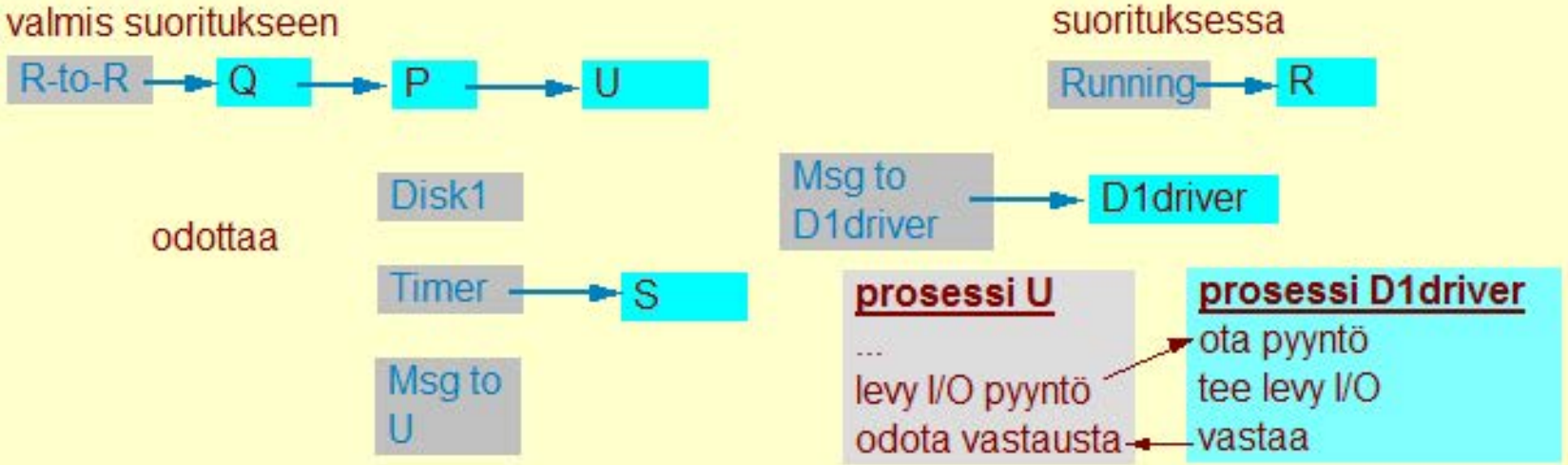
D1driver saa työn valmiiksi, U pääsee suoritusjonoon

- D1driver jää odottamaan uutta työtä, ja R pääsee suoritukseen

Copyright Teemu Kerola 2005

D1driver tutkii tilannetta ja havaitsee, että sen U:lta saama tehtävä on valmis. D1driver muotoilee viestin U:lle ja lähettää sen U:lle. Käyttöjärjestelmän prosessien välisen viestinnän toteuttava osa toimittaa viestin U:lle ja siirtää U:n ready-to-run jonoon, josta se aikanaan pääsee suoritukseen.

Esimerkki: I/O-laitekeskeytys



Lähtötilanne: U pyytänyt D1driveria lukemaan levyltä, D1driver odottaa I/O:ta

Disk1 antaa I/O-laitekeskeytyksen

- D1driver ready-to-run jonoon isolla prioriteetilla

D1driver pääsee heti suoritukseen

- ensin P takaisin jonoon ja sitten .... D1driver pääsee suoritukseen

D1driver saa työn valmiiksi, U pääsee suoritusjonoon

- D1driver jää odottamaan uutta työtä, ja R pääsee suoritukseen

Copyright Teemu Kerola 2005

D1driver on nyt valmis ja se voi siirtyä odottamaan uutta työtä, jonka se saa viestin muodossa joltakin prosessilta joskus tulevaisuudessa. Prosessi R saa nyt suoritusvuoron.

Prosessin vaihdon toteutus

Vaihdon tekee käyttöjärjestelmärutiini suorittavan prosessin ympäristössä

- talleta vanhan prosesin suoritinympäristö muistiin tuon prosesin PCB:hen
- lataa uuden prosessin suoritinympäristö rekistereihin uuden prosessin PCB:stä
- vuoro vaihtuu sillä hetkellä, kun PC saa uuden arvon

Uuden prosessin suoritus jatkuu täsmälleen samasta kohtaa kuin mihin viime suorituskerralla jäätiin

- yleensä keskellä prosessin vaihdon toteuttavaa KJ-rutiinia
- vuoronantaja voi muokata laskennassa olevan keskeytyskäsittelijän aktivointitietueen paluuosoitetta uuden prosessin PC:n mukaiseksi, jolloin keskeytyskäsittelijästä palatessa aloitetaankin suorittamaan uuden prosessin koodia
- joissakin koneissa PC:n arvo voidaan suoraan asettaa (esim LOAD-käskyllä), jolloin suoritusvuoro vaihtuu PC:n asettamiskäskyn suorituksen yhteydessä

Copyright Teemu Kerola 2005

Prosessin vaihdon toteuttaa siis aina käyttöjärjestelmärutiini, johon suoritusvuoro voi siirtyä esimerkiksi keskeytyskäsittelijän kautta. Kun uuden prosessin tietoja ladataan suoritimen rekistereille, niin jossakin vaiheessa myös PC:n arvo vaihtuu osoittamaan uuden prosessin koodia. Tämä on juuri se hetki ja konekäsky, jolloin prosessin vaihto sitten lopulta toteutuu.

Prosessin vaihdon toteutus

Vaihdon tekee käyttöjärjestelmärutiini suorittavan prosessin ympäristössä

- talleta vanhan prosesin suoritinympäristö muistiin tuon prosesin PCB:hen
- lataa uuden prosessin suoritinympäristö rekistereihin uuden prosessin PCB:stä
- vuoro vaihtuu sillä hetkellä, kun PC saa uuden arvon

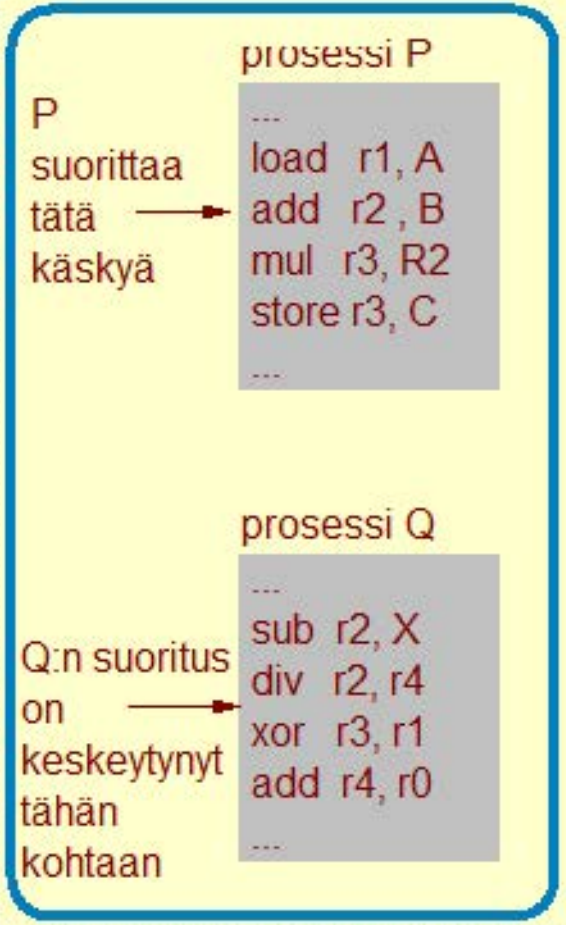
Uuden prosessin suoritus jatkuu täsmälleen samasta kohtaa kuin mihin viime suorituskerralla jäätiin

- yleensä keskellä prosessin vaihdon toteuttavaa KJ-rutiinia
- vuoronantaja voi muokata laskennassa olevan keskeytyskäsittelijän aktivointitietueen paluuosoitetta uuden prosessin PC:n mukaiseksi, jolloin keskeytyskäsittelijästä palatessa aloitetaankin suorittamaan uuden prosessin koodia
- joissakin koneissa PC:n arvo voidaan suoraan asettaa (esim LOAD-käskyllä), jolloin suoritusvuoro vaihtuu PC:n asettamiskäskyn suorituksen yhteydessä

Copyright Teemu Kerola 2005

PC:n arvon sijoittava konekäsky on yleensä keskellä prosessien hallinnan toteuttavaa käyttöjärjestelmärutiinia. Tästä seuraa myös se, että kun keskeytynyt prosessi joskus myöhemmin saa jälleen suoritusvuoron, sen suoritus jatkuu juuri tästä samasta kohtaa, keskellä käyttöjärjestelmärutiinia. Suorittava prosessi tekee rutiinin loppuun ja jatkaa sitten normaalia laskentaa. Seuraavalla sivulla on asiaa valaiseva esimerkki.

Esimerkki: prosessin vaihdon toteutus



keskeytys-käsittelijä

```
push sp, r1 ; save reg
...
svc   sp, scheduler
pop  sp, r1  ;restore r1
iret   sp
```

scheduler

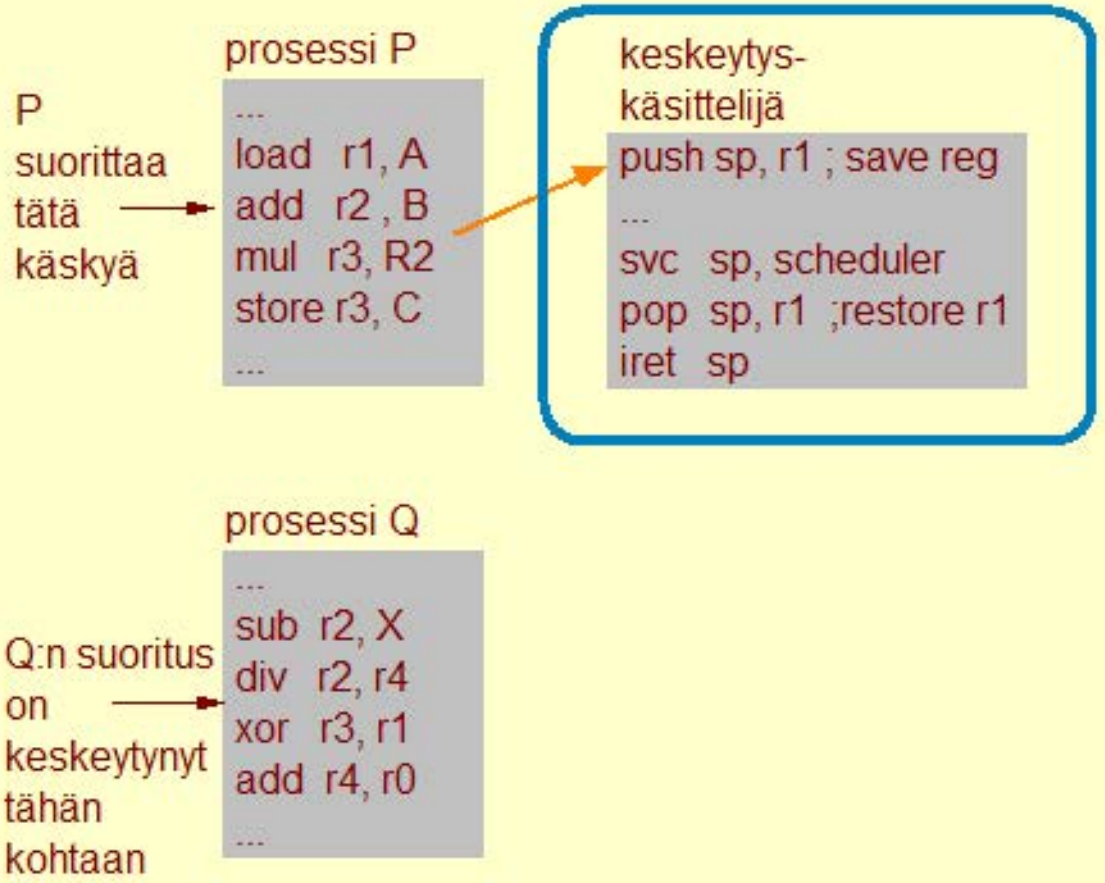
```
...
; save old CPU environment
store r0, old_r0
store r1, old_r1
...
store r7, old_r7
...

; load new CPU environment
load r0, new_r0
load r1, new_r1
...
load   r7, new_r7
set    base, new_base
set    limit, new_limit
iret    sp
```

Copyright Teemu Kerola 2005

Tässä esimerkissä meillä on kaksi prosessia, P ja Q. Prosessi Q odottaa ready-to-run -jonossa vuoroaan ja sen suoritus on keskeytynyt juuri ennen xor-käskyä. Prosessi P suorittaa ja sillä on add-käsky suorituksessa, kun tapahtuu kellolaitekeskeytys.

Esimerkki: prosessin vaihdon toteutus



scheduler

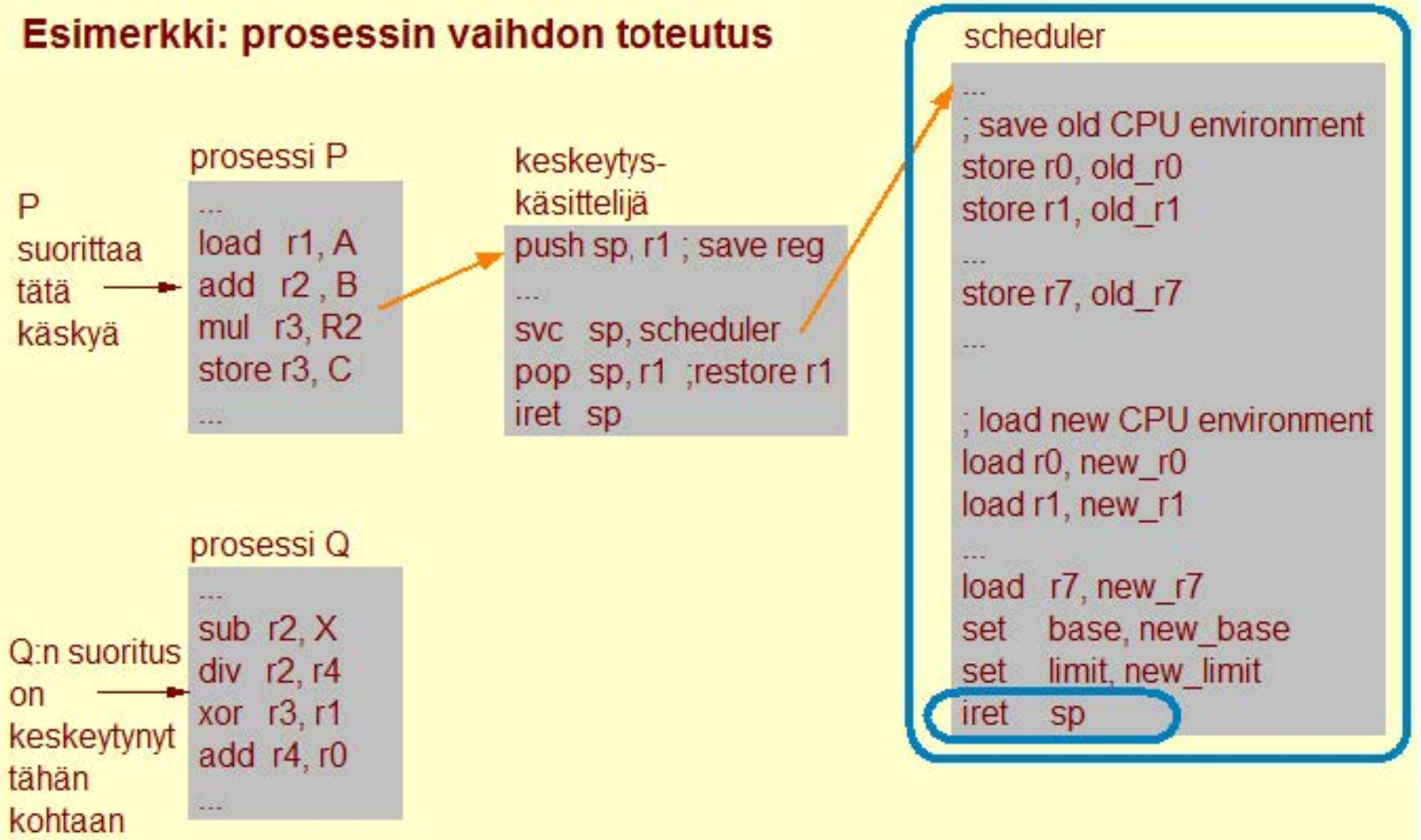
```
...
; save old CPU environment
store r0, old_r0
store r1, old_r1
...
store r7, old_r7
...

; load new CPU environment
load r0, new_r0
load r1, new_r1
...
load  r7, new_r7
set   base, new_base
set   limit, new_limit
iret  sp
```

Copyright Teemu Kerola 2005

Laitteisto havaitsee keskeytyksen ja P:n kertolaskukäskyn asemesta alkaakin suorittaa keskeytyskäsittelijää. Keskeytyskäsittelijässä talletetaan sen käyttämän rekisterin r1 nykyarvo talteen ja tehdään varsinainen työ. Käyttöjärjestelmä on keskeyttävää tyyppiä, joten lopulta keskeytyskäsittelijä kutsuu vuoronantajaa (sheduler) tarkistamaan, jos suoritin pitäisikin nyt antaa jollekin muulle prosessille kuin P:lle.

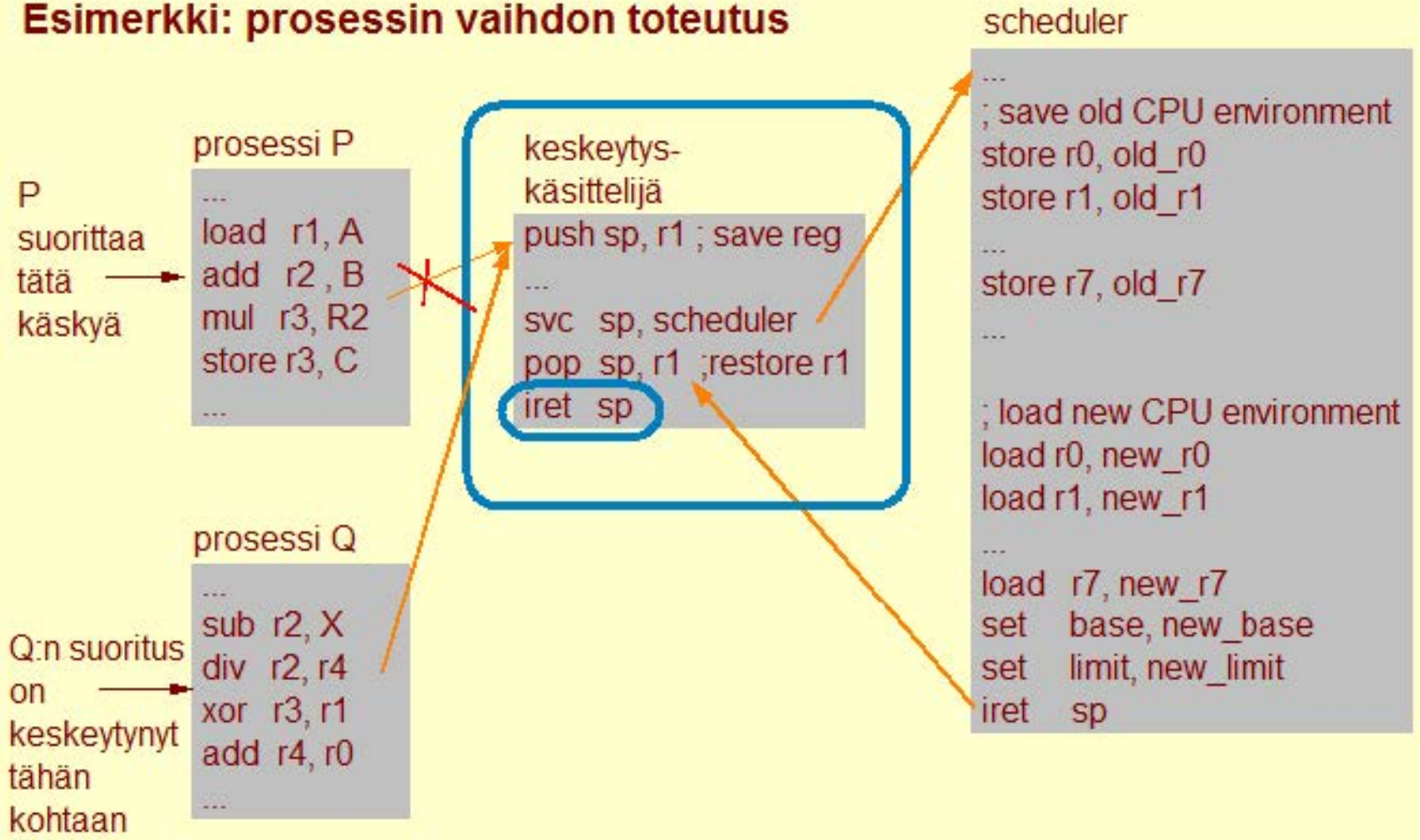
Esimerkki: prosessin vaihdon toteutus



Copyright Teemu Kerola 2005

Vuoronantaja tutkii tilannetta ja toteaa, että P joutuu luopumaan suorittimesta Q:n hyväksi. P:n suoritinympäristö talletetaan muistiin ja Q:n suoritinympäristö ladataan rekistereihin. Huomaa, että tässä esimerkissä sekä rekistereiden vanhojen arvojen talletus että niiden uusien arvojen lataus on esitetty yksinkertaistettuna. Tallealueet löytyvät oikeasti PCB:n kautta, mutta PCB:n käyttö puuttuu esimerkistä kokonaan. Esimerkissä on muitakin yksinkertaistuksia. Kun iret-käsky lopulta suoritetaan, niin schedulerin koko suoritusympäristö on juuri vaihdettu Q:n ympäristöön. Prosessin vaihto tapahtuu siis tällä kertaa tässä iret-käskyssä.

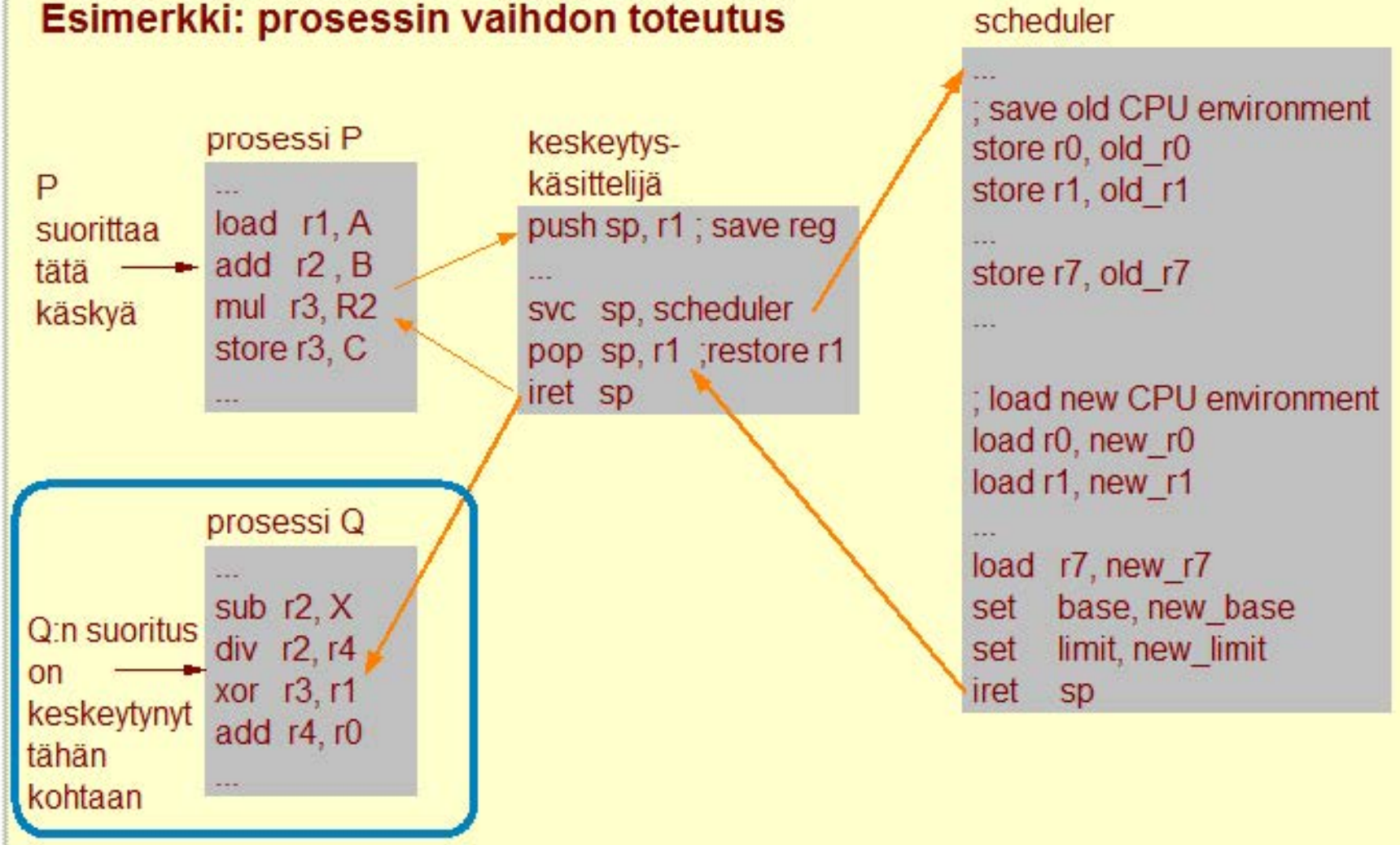
Esimerkki: prosessin vaihdon toteutus



Copyright Teemu Kerola 2005

Nyt kun keskeytyskäsittelijään palataan, niin kyseessä onkin sellainen keskeytyskäsittelijän suoritusinstassi, joka on käynnistynyt prosessin Q suorituksen keskeytymishetkellä. Se voi olla sama tai eri keskeytyskäsittelijä kuin mitä P kutsui. Tässä esimerkissä se on nyt sama keskeytyskäsittelijä kuin mitä P kutsui. Keskeytyskäsittelijässä palautetaan R1:n arvo prosessin Q ympäristöstä ja palataan keskeytyneeseen prosessiin Q iret-käskyllä.

Esimerkki: prosessin vaihdon toteutus



Copyright Teemu Kerola 2005

Iret-käsky palauttaa nyt kontrollin prosessille Q, joka jatkaa suoritustaan alkuperäisestä keskeytyskohdastaan eli xor-käskystä. Prosessi P on keskeytynyt ennen mul-käskyn suoritusta, mutta prosessin P kontrolli on todellisuudessa keskeytyskäsittelijän pop-käskyn kohdalla. Kun P aikanaan pääsee taas suoritukseen, se ensin suorittaa tuon pop-käskyn ja palaa iret-käskyllä varsinaiseen koodiinsa ja kertolaskukäskyyn. Prosessien hallinta ei siis ole ihan triviaalia.

Prosessin prioriteetti

Prosessin tärkeysjärjestys suorittimella

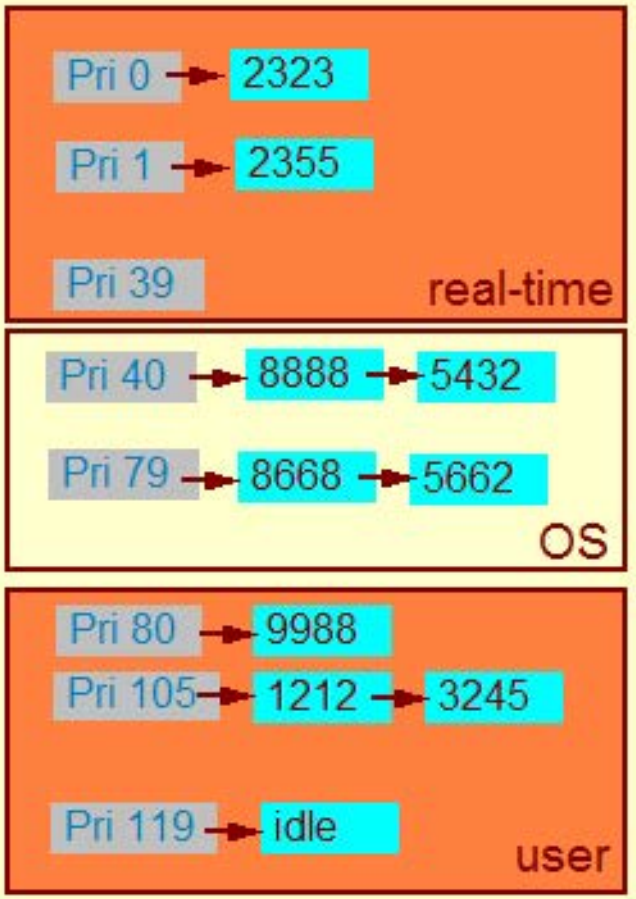
- esim. pieni numero → iso (parempi) prioriteetti

Joka prioriteettiluokalle oma jononsa

- käyttöjärjestelmäprosesseilla parempi prioriteetti kuin käyttäjätason prosesseilla
- tosiaikasovellusten prosesseilla parempi prioriteetti kuin käyttöjärjestelmäprosesseilla
  - käyttöjärjestelmälle pitää antaa aikaa toimia aina aika ajoin

Prioriteetti voi vaihdella prosessin elinaikana

- paljon suoritinaikaa → huonompi prioriteetti
- kauan ready-to-run jonossa → parempi prioriteetti
- prioriteetin vaihtelu toteutetaan siirtämällä prosessi yhden prioriteettiluokan jonosta toiseen



Copyright Teemu Kerola 2005

Prosessin prioriteetti vaikuttaa siis sen vuoronantoon suorittimella, mutta ei esimerkiksi I/O-laitteilla. Useimmissa käyttöjärjestelmissä käytetään prioriteetteja ja ne on toteutettu usealla ready-to-run -jonolla, yksi kutakin prioriteettiluokkaa kohden. Esimerkiksi Unix- ja Linux-järjestelmissä prosessin suuri prioriteettiluku tarkoittaa, että prosessi on vähemmän tärkeä.

Prosessin prioriteetti

Prosessin tärkeysjärjestys suorittimella

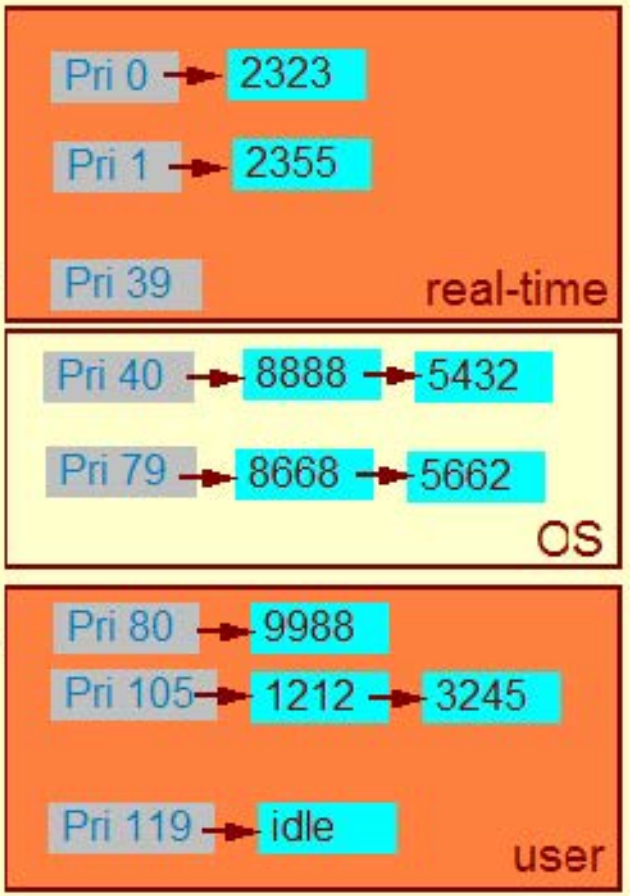
- esim. pieni numero → iso (parempi) prioriteetti

Joka prioriteettiluokalle oma jononsa

- käyttöjärjestelmäprosesseilla parempi prioriteetti kuin käyttäjätason prosesseilla
- tosiaikasovellusten prosesseilla parempi prioriteetti kuin käyttöjärjestelmäprosesseilla
  - käyttöjärjestelmälle pitää antaa aikaa toimia aina aika ajoin

Prioriteetti voi vaihdella prosessin elinaikana

- paljon suoritinaikaa → huonompi prioriteetti
- kauan ready-to-run jonossa → parempi prioriteetti
- prioriteetin vaihtelu toteutetaan siirtämällä prosessi yhden prioriteettiluokan jonosta toiseen



Copyright Teemu Kerola 2005

Prosessit voidaaan jakaa esimerkiksi kahteen tai kolmeen prioriteettikastiin, ja kussakin kastissa vielä useaan prioriteettiluokkaan. Tällä tavoin esimerkiksi käyttöjärjestelmäprosesseille on tarjolla usea prioriteetti, mutta ne ovat kaikki parempia kuin käyttäjätason prosessien prioriteetit ja samalla kertaa huonompia kuin tosiaikasovellusten prioriteetit. Tosiaikasovellusten tekemisessä yhtenä vaikeutena on suunnitella sovellus siten, että huonomman prioriteetin omaavat käyttöjärjestelmäprosessit saavat kuitenkin tarpeeksi suoritinaikaa järjestelmän hallinnan ylläpitämiseksi.

Prosessin prioriteetti

Prosessin tärkeysjärjestys suorittimella

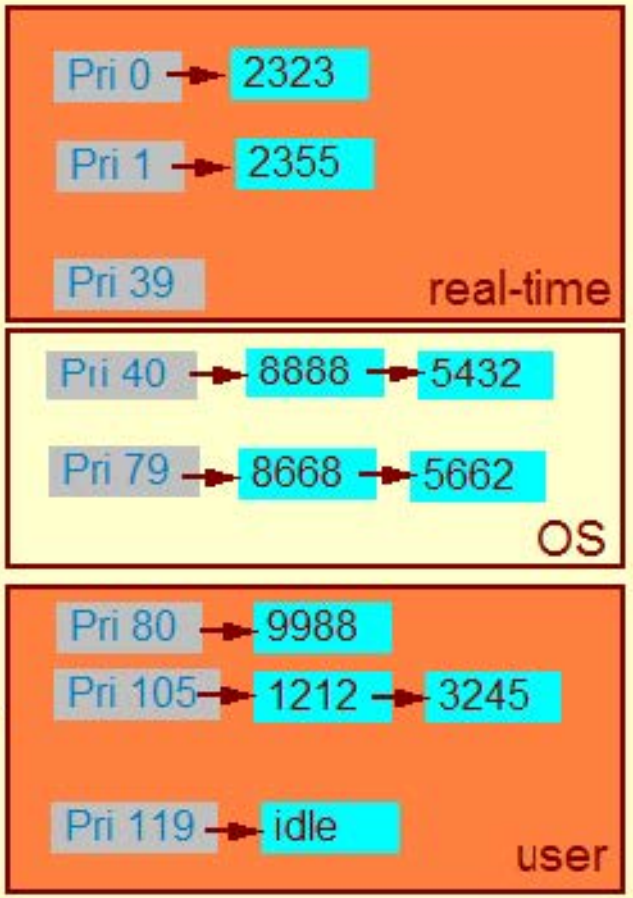
- esim. pieni numero → iso (parempi) prioriteetti

Joka prioriteettiluokalle oma jononsa

- käyttöjärjestelmäprosesseilla parempi prioriteetti kuin käyttäjätason prosesseilla
- tosiaikasovellusten prosesseilla parempi prioriteetti kuin käyttöjärjestelmäprosesseilla
  - käyttöjärjestelmälle pitää antaa aikaa toimia aina aika ajoin

Prioriteetti voi vaihdella prosessin elinaikana

- paljon suoritinaikaa → huonompi prioriteetti
- kauan ready-to-run jonossa → parempi prioriteetti
- prioriteetin vaihtelu toteutetaan siirtämällä prosessi yhden prioriteettiluokan jonosta toiseen



Copyright Teemu Kerola 2005

Prosessin prioriteetti voi myös vaihdella prosessin elinaikana useissa järjestelmissä. Tämä auttaa 'reilun pelin' toteuttamisessa suoritinajan jaossa. Jos prosessi käyttää paljon suoritinaikaa, niin sen prioriteetti heikkenee eli se viedään seuraavan kerran huonomman prioriteetin ready-to-run -jonoon. Vastaavasti kauan aikaa jonottaneiden prosessien prioriteetti kasvaa pikkuhiljaa, jolloin niiden mahdollisuudet saada suoritusvuoro tulee paremmaksi koko ajan. Missään tapauksesssa esimerkiksi käyttäjätason prosessien prioriteetti ei kuitenkaan voi kasvaa yhtä suureksi kuin huonoimmankaan prioriteetin omaavan käyttöjärjestelmäprosessin prioriteetti.

Käyttöjärjestelmä käyttäjän näkökulmasta

hidas, tahmea?

verkko tökkii?

liikkuva kuva nykii?

Miten järjestelmä toimii minun ohjelmani kanssa?

Onko järjestelmä riittävän nopea pyörittämään suosikkipeliäni kahdella isolla näytöllä?

Voinko säätää käyttöjärjestelmää siten, että pelini toimisi nopeammin?

Onko minun helppo asentaa uusi ohjelma koneelle?

Onko minun helppo asentaa uusi näyttö koneelle?

Onko minun helppo muuntaa (portata) tekemäni ohjelma tähän käyttöjärjestelmään toisesta käyttöjärjestelmästä?

Copyright Teemu Kerola 2005

Käyttäjän näkökulmasta käyttöjärjestelmä on parhaimmillaan silloin kun sitä ei huomaa lainkaan. Hyvin toimivaa systeemiä ei useinkaan osaa tarpeeksi arvostaa, mutta huonosti toimivan järjestelmän kyllä huomaa heti. Asian tekee vielä sekavaksi se, että käyttäjillä ei useinkaan ole selvää mielikuvaa siitä, mitä oikeastaan kuuluu käyttöjärjestelmään ja mitä ei. Tiedostojen hallinta selvästi mielletään käyttöjärjestelmän osaksi, mutta verkkoselaimen tai ikkunointiin perustuva käyttöliittymän osalta tilanne on epäselvempi. Yhtenä selvänä piirteenä käyttöjärjestelmälle on, että toimiiko se hyvin jonkin tietyn ohjelman kanssa vai hidastaako se turhaa ohjelman suoritusta?

Käyttöjärjestelmä käyttäjän näkökulmasta

Miten järjestelmä toimii minun ohjelmani kanssa?

Onko järjestelmä riittävän nopea pyörittämään suosikkipeliäni kahdella isolla näytöllä?

Voinko säätää käyttöjärjestelmää siten, että pelini toimisi nopeammin?

Onko minun helppo asentaa uusi ohjelma koneelle?

Onko minun helppo asentaa uusi näyttö koneelle?

Onko minun helppo muuntaa (portata) tekemäni ohjelma tähän käyttöjärjestelmään toisesta käyttöjärjestelmästä?

huonolla resoluutiolla ok, mutta hyvällä resoluutiolla tökkii?

kännykässä on liian vähän ohjelmamuistia javasovelluksille?

Copyright Teemu Kerola 2005

Käyttäjän näkökulmasta käyttöjärjestelmä voi olla hidaste ohjelmille. Joskus sanotaan, että jokin tietty ohjelma toimii hitaammin Windowsissa kuin Playstationilla, vaikka todellisuudessa kyseessä voi olla vain halvalla ja huonosti toteutettu Windows implementaatio. Toisaalta pitää myös paikkansa, että kaikki käyttöjärjestelmät eivät ole soveliaita kaikkiin tarkoituksiin. Esimerkiksi, kännykän käyttöjärjestelmälle asetetaan ihan erilaisia vaatimuksia kuin ydivoimalan kontrollijärjestelmän käyttöjärjestelmälle. Loppujen lopuksi kyse on kuitenkin siitä, soveltuuko tietty käyttöjärjestelmä tietylle sovellukselle.

Käyttöjärjestelmä käyttäjän näkökulmasta

Miten järjestelmä toimii minun ohjelmani kanssa?

Onko järjestelmä riittävän nopea pyörittämään suosikkipeliäni kahdella isolla näytöllä?

Voinko säätää käyttöjärjestelmää siten, että pelini toimisi nopeammin?

Onko minun helppo asentaa uusi ohjelma koneelle?

Onko minun helppo asentaa uusi näyttö koneelle?

Onko minun helppo muuntaa (portata) tekemäni ohjelma tähän käyttöjärjestelmään toisesta käyttöjärjestelmästä?

Epäselviä skriptikomentoja, joissa on usea parametri?

Hieno graafinen käyttöliittymä turvatarkistuksien kera?

Oops - nyt se toimi enää lainkaan! Mitenkä pääsen takaisin edelliseen tilaan?

Copyright Teemu Kerola 2005

Käyttäjä ei useinkaan halua millään tavalla säätää käyttöjärjestelmää - itse asiassa useimmat käyttäjät eivät halua edes tietää käyttöjärjestelmän olemassaolosta mitään. Valistuneet käyttäjät, kuten esimerkiksi sinä, voivat kuitenkin joskus haluta muokata käyttöjärjestelmän parametreja, jotta se toimisi paremmin. Eri käyttöjärjestelmillä on huomattavasti erilaisia käyttäjän rajapintoja parametrien muokkaamiseen. Hyvissä käyttöjärjestelmissä tällaiset rajapinnat ovat selkeitä ja ne samalla turvaavat käyttäjää tekemästä suuria virheitä, jotka esimerkiksi tekisivät koko järjestelmän toimimattomaksi.

Käyttöjärjestelmä käyttäjän näkökulmasta

autoinstall?

vain ammattilaisille?

Miten järjestelmä toimii minun ohjelmani kanssa?

Miten tehdä niin, että Jukka ja Maija saavat käyttää, mutta Liisa ei?

Onko järjestelmä riittävän nopea pyörittämään suosikkipeliäni kahdella isolla näytöllä?

Asenna niin, että tilapäiset isot suoritusaikaiset tiedostot talletetaan partitioon D hakemistoon tmp?

Voinko säätää käyttöjärjestelmää siten, että pelini toimisi nopeammin?

Onko minun helppo asentaa uusi ohjelma koneelle?

Onko minun helppo asentaa uusi näyttö koneelle?

Onko minun helppo muuntaa (portata) tekemäni ohjelma tähän käyttöjärjestelmään toisesta käyttöjärjestelmästä?

Copyright Teemu Kerola 2005

Uusien ohjelmien asennus järjestelmään on aika yleinen toimenpide. Useinhan riittää, että asennusohjelma voidaan suorittaa esimerkiksi CD- tai DVD-levyltä, jolloin käyttöjärjestelmä voi käynnistää asennuksen automaattisesti levyn levyasemaan sijoittamisen yhteydessä. Isommissa järjestelmissä tulisi olla helppo asentaa ohjelma useillakin erilaisilla käyttötavoilla ja käyttöjärjestelmän tulisi tukea tai ainakin sallia tällaiset asennukset.

Käyttöjärjestelmä käyttäjän näkökulmasta

Miten järjestelmä toimii minun ohjelmani kanssa?

Onko järjestelmä riittävän nopea pyörittämään suosikkipeliäni kahdella isolla näytöllä?

Voinko säätää käyttöjärjestelmää siten, että pelini toimisi nopeammin?

plug-and-play?

vain ammattilaisille?

Onko minun helppo asentaa uusi ohjelma koneelle?

mistä ohjelmistopäivitykset?

Onko minun helppo asentaa uusi näyttö koneelle?

Onko minun helppo muuntaa (portata) tekemäni ohjelma tähän käyttöjärjestelmään toisesta käyttöjärjestelmästä?

Copyright Teemu Kerola 2005

Käyttöjärjestelmän tulisi myös tehdä helpoksi uusien laitteiden liittämisen järjestelmään. Esimerkiksi Linux'in laajempaa käyttöönottoa on huomattavasti haitannut mielikuva, että järjestelmän ylläpito on huomattavasti vaikeampaa kuin Windows'in. Uusien laitteiden laiteajurien asennuksen tulisi olla helppoa. Olisi mukavaa, jos sen voisi tehdä useimpien laitteiden osalta siten, että järjestelmää ei tarvitsisi bootata asennuksen jälkeen.

Käyttöjärjestelmä käyttäjän näkökulmasta

Miten järjestelmä toimii minun ohjelmani kanssa?

Onko järjestelmä riittävän nopea pyörittämään suosikkipeliäni kahdella isolla näytöllä?

Voinko säätää käyttöjärjestelmää siten, että pelini toimisi nopeammin?

Onko minun helppo asentaa uusi ohjelma koneelle?

Onko minun helppo asentaa uusi näyttö koneelle?

Onko minun helppo muuntaa (portata) tekemäni ohjelma tähän käyttöjärjestelmään toisesta käyttöjärjestelmästä?

Riittääkö uudelleen käännös?

Löytyykö standardin mukainen Java-kirjasto?

Voiko ohjelmakirjastoja linkittää dynaamisesti suoritusaikana?

Copyright Teemu Kerola 2005

Ohjelmistojen kehittäjät ovat myös järjestelmän käyttäjiä, vaikkakin aika lailla enemmän asiantuntemusta omaavia kuin tavalliset pulliaiset. Käyttöjärjestelmässä olisi suotavaa olla selkeästi määritellyt rajapinnat, joiden avulla mikä tahansa ohjelma olisi helppo alkuaan rakentaa tai myöhemmin portata toisesta käyttöjärjestelmästä tähän käyttöjärjestelmään. Rajapintojen tulisi olla julkisia ja staattisia siten, että ne eivät muutu käyttöjärjestelmän seuraavan version yhteydessä.

Ylläpitäjän näkökulma käyttöjärjestelmään

Ovatko kaikki systeemin resurssit mahdollisimman hyvässä käytössä?

Mikä on keskimääräinen ready-to-run jonon pituus eli suoritinta odottavien prosessien lukumäärä?

Minkä osan ajasta suoritin vain odottaa järkevää työtä?

Minkä osan ajasta kovalevyn hakuvarsi on liikkeessä?

Miten usein datamuistiviitteet löytyvät välimuistista?

Miten muistin lisääminen vaikuttaisi nopeuteen?
Miten grafiikkakortin muistin lisääminen vaikuttaisi nopeuteen?

Tasapainoinen systeemi?

Tekstinkäsittelyyn käytettävässä läppärissä 4GB muistia tai 3 GHz suoritin?

Kuka tuhlasi rahaa hienoon Mersuun, kun perus-Toyota olisi riittänyt?

Onhan virtuaalimuistille sopivasti levytilaa?

Onhan tiedostovälimuisti riittävän iso?

Copyright Teemu Kerola 2005

Järjestelmän ylläpitäjän näkökulma käyttöjärjestelmään on aivan erilainen. Hän on lähinnä kiinnostunut siitä, kuinka hyvin järjestelmä kokonaisuutena toimii, eikä siitä, miten hyvin se toimii juuri yhden tietyn sovelluksen kanssa. Hyvä laitteisto on tasapainoinen, jossa kaikki osaset ovat suunnilleen samaa tasoa eikä mikään ole huomattavasti hitaampi tai nopeampi kuin muut. Järjestelmän kaikkien osien tulisi olla tehokkaassa käytössä - muutenhan laitteisto on turhan hieno ja siis kallis käyttötarkoitukseensa. Toisaalta, uuden laitteiston alikäyttö on OK, kunhan sitten noin kolmen vuoden päästä se on tehokkaassa käytössä.

Ylläpitäjän näkökulma käyttöjärjestelmään

Ovatko kaikki systeemin resurssit mahdollisimman hyvässä käytössä?

Onko suoritin tarpeeksi nopea?

Mikä on keskimääräinen ready-to-run jonon pituus eli suoritinta odottavien prosessien lukumäärä?

Onko suorittimia tarpeeksi monta?

Minkä osan ajasta suoritin vain odottaa järkevää työtä?

Minkä osan ajasta kovalevyn hakuvarsi on liikkeessä?

Miten usein datamuistiviitteet löytyvät välimuistista?

Miten muistin lisääminen vaikuttaisi nopeuteen?
Miten grafiikkakortin muistin lisääminen vaikuttaisi nopeuteen?

Copyright Teemu Kerola 2005

Ylläpitäjä voi aika ajoin tutkia esimerkiksi web-serverin ready-to-run -jonon pituutta. Jos jonon pituus on hyvin suuri, tämä voi olla indikaatio laskentatehon puutteesta. Toisaalta, se voi myös merkitä vain sitä, että laitteisto tekee paljon töitä!. Jos web-töiden vasteajat ovat silti kohtuullisia, niin järjestelmä on vain tehokkaassa käytössä, vaikkakin ehkä lähellä kapasiteettiaan.

Ylläpitäjän näkökulma käyttöjärjestelmään

Onko suoritin liian nopea?

Onko järjestelmä turhan tehokas?

Mikä on tilanne vuoden päästä?

Ovatko kaikki systeemin resurssit mahdollisimman hyvässä käytössä?

Mikä on keskimääräinen ready-to-run jonon pituus eli suoritinta odottavien prosessien lukumäärä?

Minkä osan ajasta suoritin vain odottaa järkevää työtä?

Minkä osan ajasta kovalevyn hakuvarsi on liikkeessä?

Miten usein datamuistiviitteet löytyvät välimuistista?

Miten muistin lisääminen vaikuttaisi nopeuteen?
Miten grafiikkakortin muistin lisääminen vaikuttaisi nopeuteen?

Copyright Teemu Kerola 2005

Jos suoritin on suuren osan ajasta tekemättä mitään, niin se taas voi olla indikaatio turhan tehokkaasta suorittimesta. Toisaalta, voi olla, että palvelun tarpeen otaksutaan kasvavan lähiaikoina paljon, joten on hyväkin olla extra-kapasiteettia valmiina.

Ylläpitäjän näkökulma käyttöjärjestelmään

Ovatko kaikki systeemin resurssit mahdollisimman hyvässä käytössä?

Onko kovalevy tarpeeksi nopea?

Mikä on keskimääräinen ready-to-run jonon pituus eli suoritinta odottavien prosessien lukumäärä?

Tarvitaanko lisää kovalevyjä?

Minkä osan ajasta suoritin vain odottaa järkevää työtä?

Minkä osan ajasta kovalevyn hakuvarsi on liikkeessä?

Onko käytetty RAID-systeemi sopiva?

Miten usein datamuistiviitteet löytyvät välimuistista?

Miten muistin lisääminen vaikuttaisi nopeuteen?
Miten grafiikkakortin muistin lisääminen vaikuttaisi nopeuteen?

Copyright Teemu Kerola 2005

I/O-systeemin kapasiteettia voi estimoida jollain tarkkudella tutkimalla hakuvarren liikettä. Jos hakuvarsi liikkuu koko ajan, se viittaa levyn kovaan käyttöasteeseen, mikä taas sovelluksesta riippuen voi olla joko hyvä tai huono asia. Jos sovellus ei tee juuri mitään levy-I/O:ta, mutta hakuvarsi sahaa koko ajan edes takaisin, kyseessä voi olla virtuaalimuistin tukimuistiin liittyvä toiminto. Tämä taas voi aiheutua joko riittämättömän pienestä tukimuistin levypartitiosta tai sitten liian suuresta määrästä samaan aikaan suorituksessa olevia prosesseja.

Ylläpitäjän näkökulma käyttöjärjestelmään

Ovatko kaikki systeemin resurssit mahdollisimman hyvässä käytössä?

Mikä on keskimääräinen ready-to-run jonon pituus eli suoritinta odottavien prosessien lukumäärä?

Minkä osan ajasta suoritin vain odottaa järkevää työtä?

Minkä osan ajasta kovalevyn hakuvarsi on liikkeessä?

Miten usein datamuistiviitteet löytyvät välimuistista?

Miten muistin lisääminen vaikuttaisi nopeuteen?
Miten grafiikkakortin muistin lisääminen vaikuttaisi nopeuteen?

Onko ohjelma käännetty siten, että se osaa hyödyntää tämän järjestelmän välimuistia?

Onko meillä hyvät kääntäjät?

Onko suorittimella tarpeeksi välimuistia?

Copyright Teemu Kerola 2005

Vaikka välimuisti onkin yleensä kiinteä osa suoritinta, niin käyttöjärjestelmä voi vaikuttaa sen tehokkaaseen käyttöön. Kääntäjien pitäisi pystyä optimoimaan koodia siten, että sovellus toimii tehokkaasti juuri tämän välimuistin kanssa.

Ylläpitäjän näkökulma käyttöjärjestelmään

Ovatko kaikki systeemin resurssit mahdollisimman hyvässä käytössä?

Missä on järjestelmän suorituskykypullonkaula?

Mikä on keskimääräinen ready-to-run jonon pituus eli suoritinta odottavien prosessien lukumäärä?

Montako vuotta tämä järjestelmä on vielä riittävän nopea?

Minkä osan ajasta suoritin vain odottaa järkevää työtä?

Minkä osan ajasta kovalevyn hakuvarsi on liikkeessä?

Milloin järjestelmä pitää uusia?

Miten usein datamuistiviitteet löytyvät välimuistista?

Riittäisikö grafiikka-kortin uusiminen?

Miten muistin lisääminen vaikuttaisi nopeuteen? Miten grafiikkakortin muistin lisääminen vaikuttaisi nopeuteen?

Copyright Teemu Kerola 2005

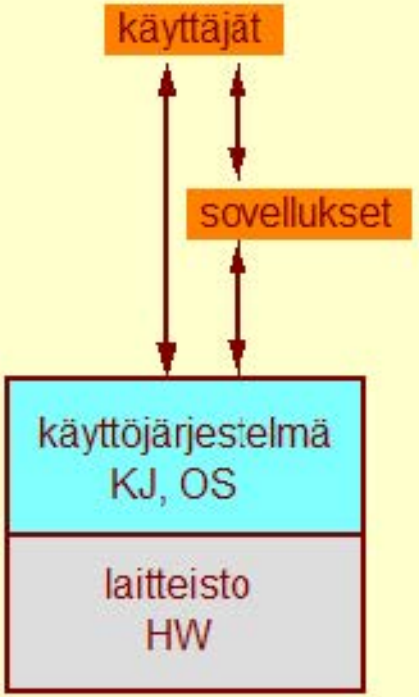
Ylläpitäjä myös suunnittelee järjestelmän laajennukset ja käyttöjärjestelmän tulisi antaa hänelle riittävästi tietoa järkevien päätösten tekemiseen. Järjestelmän hallinta on usein yllättävän monimutkaista ja edelleenkin tapahtuu liian usein niin, että kotikäyttäjä sekavien päivitysten jälkeen aloittaa alusta eli installoi koko käyttöjärjestelmän uudelleen järjestelmän vakauttamiseksi.

Käyttöjärjestelmä käyttöliittymänä laitteistoon

Loppukäyttäjälle
- ihmiselle

Sovellusohjelmille
- ohjelmiston kehitysaikana
- ohjelmiston suoritusaikana

Piilottaa laitteiston erityispiirteet käyttäjiltä
- käskykanta, konekäskyjen rakenne
- suorittimen toteutus, suorittimien lukumäärä
- I/O:n toteutus, I/O-laitteiden tyyppi, valmistaja, merkki, versio
- I/O-laitteiden sijainti
  - oma kone
  - Intranetin verkkopalvelin
  - Internetin verkkopalvelin



Copyright Teemu Kerola 2005

Käyttöjärjestelmän voi myös ajatella olevan puhdas käyttöliittymä laitteistoon. Tämä tarkoittaa sitä, että tietokoneen laitteistoa ei koskaan käsitellä suoraan, esimerkiksi kääntämällä jotain säätöruuvia ruuvimeisselillä, vaan kaikki käyttö tapahtuu käyttöjärjestelmäohjelmiston avulla. Käyttöjärjestelmä siis kaiken muun lisäksi toteuttaa myös toiminnallisen käyttöliittymän koko järjestelmään. Tämä käyttöliittymäaspekti käyttöjärjestelmissä on aika uusi ja se on kunnolla tiedostettu vasta muutaman vuosikymmenen. Mutta juuri se on ollut merkittävä tekijä esimerkiksi Unixin (ja siis myös Linuxin) sekä Windowsin leviämiseen. Onhan Windows-järjestelmän nimikin tullut nimenomaan sen käyttöliittymän perusteella.

Käyttöjärjestelmä käyttöliittymänä laitteistoon

Loppukäyttäjälle

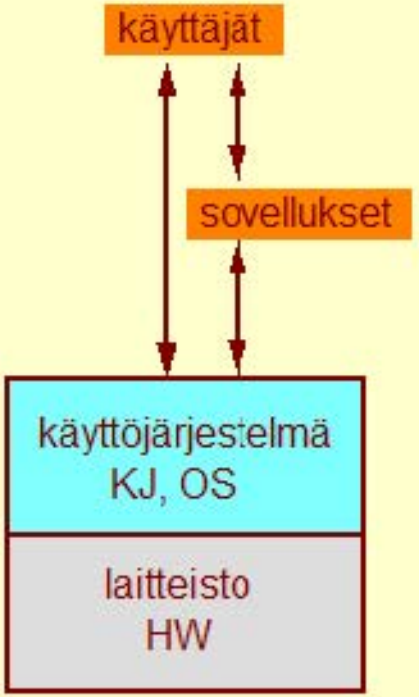
- ihmiselle

Sovellusohjelmille

- ohjelmiston kehitysaikana
- ohjelmiston suoritusaikana

Piilottaa laitteiston erityispiirteet käyttäjiltä

- käskykanta, konekäskyjen rakenne
- suorittimen toteutus, suorittimien lukumäärä
- I/O:n toteutus, I/O-laitteiden tyyppi, valmistaja, merkki, versio
- I/O-laitteiden sijainti
  - oma kone
  - Intranetin verkkopalvelin
  - Internetin verkkopalvelin



Copyright Teemu Kerola 2005

Myös sovellusohjelmat ovat järjestelmän käyttäjiä. Toimivathan sovellukset suoritusaikanaan autonomisesti ja käyttävät kaikkia laitteiston resursseja. Sovellukset toteutetaan nimenomaan käyttöjärjestelmän antaman rajapinnan päälle, eikä millekään tietylle laitteistolle. Vastaavasti suoritusaikana sovellukset käyttävät laitteistoa ainoastaan käyttöjärjestelmän kautta.

Käyttöjärjestelmä käyttöliittymänä laitteistoon

Loppukäyttäjälle

- ihmiselle

Sovellusohjelmille

- ohjelmiston kehitysaikana
- ohjelmiston suoritusaikana

Piilottaa laitteiston erityispiirteet käyttäjiltä

- käskykanta, konekäskyjen rakenne
- suorittimen toteutus, suorittimien lukumäärä
- I/O:n toteutus, I/O-laitteiden tyyppi, valmistaja, merkki, versio
- I/O-laitteiden sijainti
  - oma kone
  - Intranetin verkkopalvelin
  - Internetin verkkopalvelin

Copyright Teemu Kerola 2005

Käyttöjärjestelmä itse asiassa piilottaa laitteiston yksityiskohdat käyttäjiltä. Sovellusten kehittäjät eivät yleensä edes voi päästä käsiksi itse laitteistoon, jolloin sovelluksista tulee luontevasti laitteistoriippumattomia. Käyttäjien ja sovellusten ei yleensä tarvitse tietää, minkälainen laitteisto on käytössä, mikä on konekäskykanta tai montako suoritinta järjestelmässä on. Ohjelmistot toimivat kaikkien I/O-laitteiden kanssa ja useat ohjelmistot jopa niin, että I/O-laitteen sijaintikin voidaan määritellä vasta suoritusaikana ja käyttöjärjestelmä toteuttaa tämän kaiken.

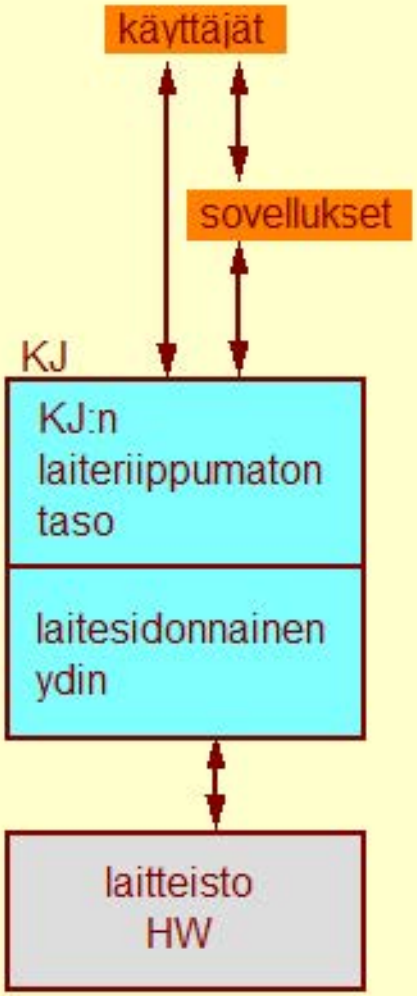
Käyttöjärjestelmän tavoitteet

Laiteriippumaton käyttöliittymä laitteistoon

- järjestelmää on helppo käyttää
- järjestelmä antaa reilua palvelua kaikille
- sovellukset on helppo tehdä
- sovellukset on helppo siirtää muihin järjestelmiin ja muista järjestelmistä

Järjestelmän resurssien tehokas hallinta

- kaikista resursseista saadaan maksimihyöty
  - kuka osti liian tehokkaan levypalvelimen?
- joustava resurssien yhteiskäyttö
  - lue tiedosto levyltä tai verkkopalvelimelta
- tiukka tietosuoja
  - kuka muu kuin Pekka ja Maija luki tietojani?
  - Matti saa kirjoittaa, Ville lukea, mutta Tuuli ei saa koskea lainkaan?



Copyright Teemu Kerola 2005

Käyttöjärjestelmällä on siis kaksi hyvin erilaista tavoitetta. Se ensinnäkin antaa laiteriippumattoman käyttöliittymän laitteistoon, jolloin kaikkia järjestelmä voidaan yhden käyttöjärjestelmän puitteissa käyttää samalla tavalla. Käyttöjärjestelmä on usein jopa toteutettu niin, että suurin osa sitä on täysin laiteriippumatonta, ja ainoastaan pieni mutta tärkeä ydinosa on laitteistosidonnaista. Tällä tavoin myös käyttöjärjestelmän siirto uudelle laitteistolle on vähän helpompaa. On myös mukavaa, jos sama Linux-ohjelma toimii sellaisenaan sekä IBM- että Intel-pohjaisissa järjestelmissä.

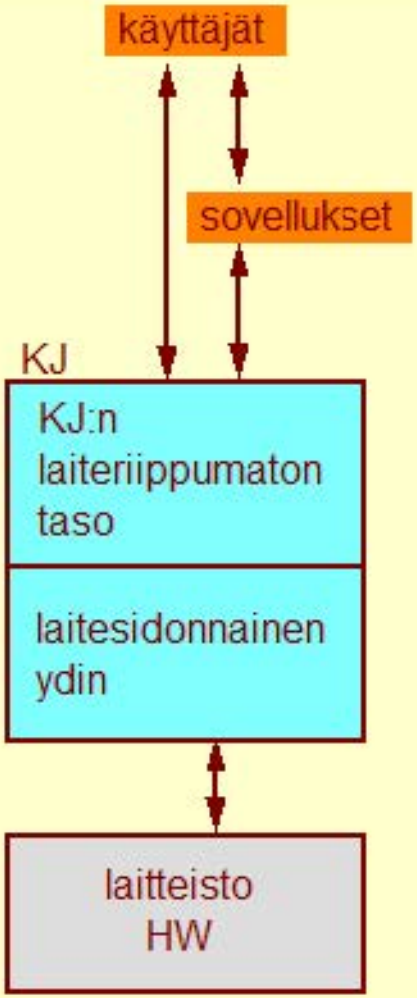
Käyttöjärjestelmän tavoitteet

Laiteriippumaton käyttöliittymä laitteistoon

- järjestelmää on helppo käyttää
- järjestelmä antaa reilua palvelua kaikille
- sovellukset on helppo tehdä
- sovellukset on helppo siirtää muihin järjestelmiin ja muista järjestelmistä

Järjestelmän resurssien tehokas hallinta

- kaikista resursseista saadaan maksimihyöty
  - kuka osti liian tehokkaan levypalvelimen?
- joustava resurssien yhteiskäyttö
  - lue tiedosto levyltä tai verkkopalvelimelta
- tiukka tietosuoja
  - kuka muu kuin Pekka ja Maija luki tietojani?
  - Matti saa kirjoittaa, Ville lukea, mutta Tuuli ei saa koskea lainkaan?



Copyright Teemu Kerola 2005

Järjestelmän ylläpitäjän näkökulmasta taas käyttöjärjestelmä huolehtii nimenomaan tämän laitteiston kaikista resursseista. Käyttöjärjestelmä pystyy hyödyntämään kaikki laitteistot maksimiteholla. Se myös tekee sopivan kompromissin kahden sinällään ristiriitaisen tavoitteen kanssa. Tottakai haluamme pystyä yhteiskäyttämään useitakin laitteiston resursseja kuten tiedostojärjestelmää ja muistia, mutta toisaalta sen pitää myös pystyä toteuttamaan riittävän tasoinen ja helppokäyttöinen tietosuoja eri käyttäjäryhmien välillä. Täytyy olla helppoa määritellä yhteiskäyttöisen tiedon suojaus hyvinkin moninaisilla käyttötavoilla.

Käyttöjärjestelmä resurssien valvojana

Suoritin

- kukaan ei odota suoritinaikaa ikuisesti, reilu peli
- kriittiset prosessit saavat ajoissa suoritinaikaa
  - KJ-prosessit? realiaikaprosessit?

Web-palvelimen vasteajat kohtuullisia kaikille asiakkaille?

Ydinvoimalan valvontajärjestelmä reagoi riittävän nopeasti veden pinnan laskuun?

Muisti

- kaikille prosesseille muistia riittävästi, mutta reilusti
- helppo yhteiskäyttö ja tietojen suojaus

Tiedostojärjestelmä

- laitteesta ja sijainnista riippumaton käyttö
- helppo yhteiskäyttö ja tietojen suojaus

Tietoliikenneverkot

- media- ja laiteriippumattomuus
- helppo yhteiskäyttö ja tietojen suojaus

Copyright Teemu Kerola 2005

Tietokonejärjestelmässä on fyysisiä resursseja yleensä neljää eri perustyyppiä. Suoritin on erikoisasemassa tällaisena resurssina, koska prosessilla pitää olla se hallussaan ennen kuin se voi tehdä yhtään mitään. Suorittimen vuoronanto ei ole ollenkaan triviaali asia ja se toteutetaan eri järjestelmissä hyvinkin eri tavoin. Esimerkiksi, moniprosessorikoneet ja realiaikaympäristöt asettavat omia vaatimuksiaan reilun ja tehokkaan vuoronannon toteuttamiseen. Näistä puhutaan sitten tarkemmin käyttöjärjestelmäkurssilla.

Käyttöjärjestelmä resurssien valvojana

Suoritin

- kukaan ei odota suoritinaikaa ikuisesti, reilu peli
- kriittiset prosessit saavat ajoissa suoritinaikaa
  - KJ-prosessit? realiaikaprosessit?

Säädä Web-palvelimen moniajoaste siten, että kaikille töille on riittävästi muistia.

Muisti

- kaikille prosesseille muistia riittävästi, mutta reilusti
- helppo yhteiskäyttö ja tietojen suojaus

Prosessit A, B ja C käyttävät yhteistä keskusmuistitietokantaa Data, jonka tietoja prosessi F ei saa lukea.

Tiedostojärjestelmä

- laitteesta ja sijainnista riippumaton käyttö
- helppo yhteiskäyttö ja tietojen suojaus

Samanaikaisuuden hallinta: entä jos kaksi prosessia yrittää kirjoittaa samaan aikaan samaan muistialueeseen?

Tietoliikenneverkot

- media- ja laiteriippumattomuus
- helppo yhteiskäyttö ja tietojen suojaus

Copyright Teemu Kerola 2005

Toinen tärkeä laitteistoresurssi on muisti. Suorituksessa olevalla prosessilla tulee olla tarpeeksi muistitilaa käytössään. Muistitilaa on kuitenkin yleensä aina rajoitetusti saatavilla, joten sitä ei pitäisi antaa liikaa millekään prosessille. Samaa muistia käytetään myös puskuralueena sekä tiedostojen että tietoliikenteen toteuttamisessa, mikä asettaa vielä uusia lisärajoitteita sen käytölle. Lisäksi järjestelmän tulee sallia eri prosessien yhteisten muistialueiden käyttö, mutta samalla turvata tiedon yksityisyys tarvittaessa. Tavoitteet ovat usein ristiriitaisia ja ratkaisut aina jonkin tason kompromisseja.

Käyttöjärjestelmä resurssien valvojana

Suoritin

- kukaan ei odota suoritinaikaa ikuisesti, reilu peli
- kriittiset prosessit saavat ajoissa suoritinaikaa
  - KJ-prosessit? realiaikaprosessit?

Muisti

- kaikille prosesseille muistia riittävästi, mutta reilusti
- helppo yhteiskäyttö ja tietojen suojaus

Tiedostojärjestelmä

- laitteesta ja sijainnista riippumaton käyttö
- helppo yhteiskäyttö ja tietojen suojaus

Tietoliikenneverkot

- media- ja laiteriippumattomuus
- helppo yhteiskäyttö ja tietojen suojaus

Prosessi P lukeekin syötteet levyllä olevalta tiedostolta MyDir/MyFile sen sijaan että lukisi ne oletusarvoisesti näppäimistöltä.

Prosessit A, B ja C käyttävät yhteistä levyllä olevaa tietokantaa Data, jonka tietoja prosessi F ei saa lukea.

Samanaikaisuuden hallinta: entä jos kaksi prosessia yrittää kirjoittaa samaan aikaan samaan tiedostoon?

Copyright Teemu Kerola 2005

Tiedostot voivat sijaita erilaisilla medioilla, kuten CD-levyillä tai kovalevyillä, ja kullekin mediatyypille on vielä hyvin suuri määrä eri laitteiden valmistajia. Järjestelmän tulee kuitenkin taata yhtenäinen ja selkeä käyttörajapinta näille kaikille eri laitteille. Myös tiedostojärjestelmille pätevät nuo samat ristiriitaiset tavoitteet tiedostojen yhteiskäytön ja yksityisyyden suojan välillä.

Käyttöjärjestelmä resurssien valvojana

Suoritin

- kukaan ei odota suoritinaikaa ikuisesti, reilu peli
- kriittiset prosessit saavat ajoissa suoritinaikaa
  - KJ-prosessit? realiaikaprosessit?

Muisti

- kaikille prosesseille muistia riittävästi, mutta reilusti
- helppo yhteiskäyttö ja tietojen suojaus

Tiedostojärjestelmä

- laitteesta ja sijainnista riippumaton käyttö
- helppo yhteiskäyttö ja tietojen suojaus

Tietoliikenneverkot

- media- ja laiteriippumattomuus
- helppo yhteiskäyttö ja tietojen suojaus

Prosessi P lukeekin syötteet verkkolevyllä Server olevalta tiedostolta MyDir/MyFile sen sijaan että lukisi ne oletusarvoisesti näppäimistöltä.

Prosessit A, B ja C käyttävät yhteistä verkkolevyllä olevaa tietokantaa Data, jonka tietoja prosessi F ei saa lukea.

Sovellus kommunikoi käyttäjän käsipuhelimen kanssa joko Bluetooth tai infrapunatekniikalla, puhelimesta riippuen.

Java-sovellus J tarvitsee seuraavaksi moduulia Apua. Mistä päin verkkoa se löytyy ja voidaanko se linkittää paikalleen sovelluksen J suoritusaikana?

Copyright Teemu Kerola 2005

Yhä useammat sovellukset käyttävät verkkoa ja verkossa olevia palvelimia ihan jokapäiväisessä toiminnassa. Verkkojen käytön pitää tietenkin olla mediasta riippumatonta, joten läppärin verkko voi yhtä hyvin olla kotona langattoman paikallisverkon takana olevassa laajakaistassa tai kännykän avulla toteutetun GPRS-yhteyden takana. Verkossa olevia palvelimia pitäisi pystyä (oikeuksien salliessa) käyttämään yhtä helposti kuin oman organisaation levypalvelinta tai oman koneen DVD- tai kovalevyä.

Käyttöjärjestelmä järjestelmän eheyden turvaajana

Varauduttu kaikkii mahdollisiin virheisiin

- kaikki virheet on vaikea edes kuvitella, saati sitten varautua niihin
- vain erityisesti sallittujen toimenpiteiden suorittaminen tekee systeemistä kompelön

Sovellusohjelmat eivät voi häiritä käyttöjärjestelmää tai muita prosesseja

- tahallaan (esim. tietokonevirukset)
- vahingossa (yleisin tapaus)
- tiedon muuttaminen
- suoritinajan anastaminen

Ei kai nyt kukaan antaisi parametrina 50000 merkin merkkijonoa?

Tällä koneella saa selata verkkoa Mosaic-selaimella www.cs.helsinki.fi domainissa. Mitään tiedostoja ei voi lukea eikä tallettaa. Järjestelmä on turvallinen.

Järjestelmä ei lukkiudu tai "kaadu"

- käyttöjärjestelmän omat tietorakenteet aina (todistettavasti) eheitä
- sovellusohjelmat eivät voi koskea käyttöjärjestelmän tietorakenteisiin
- käyttäjät tai sovellusohjelmat eivät voi saada pääkäyttäjän oikeuksia ilman lupaa

Copyright Teemu Kerola 2005

Koko järjestelmän eheyden turvaaminen on yksi käyttöjärjestelmän tärkeimmistä tehtävistä. Ei ole paljoakaan hyötyä hyvin toimivasta hallinta- ja valvontamekanismeista, jos koko järjestelmä on korruptoitunut. Perusideana on, että joka sovelluksessa ja erityisesti jokaisessa käyttöjärjestelmän osasessa on varauduttu kaikkiin mahdollisiin virheisiin jo alusta pitäen. Käytännössä tämä on kuitenkin vaikeata, joten lopputulos on aina kompromissi. Järjestelmä voidaan myös rakentaa täysin need-to-know -periaatteella, jossa vain erityisesti sallitut toimenpiteet ovat laillisia. Tällaiset järjestelmät sopivat joihinkin erityistarkoituksiin, mutta ovat yleensä liian rajoittavia.

Käyttöjärjestelmä järjestelmän eheyden turvaajana

Varauduttu kaikkii mahdollisiin virheisiin

- kaikki virheet on vaikea edes kuvitella, saati sitten varautua niihin
- vain erityisesti sallittujen toimenpiteiden suorittaminen tekee systeemistä kompelön

Sovellusohjelmat eivät voi häiritä käyttöjärjestelmää tai muita prosesseja

- tahallaan (esim. tietokonevirukset)
- vahingossa (yleisin tapaus)
- tiedon muuttaminen
- suoritinajan anastaminen

Merkkijono parametrin pituustarkastus puuttuu, jolloin sen avulla välitetty pitkä merkkijono voi muuttaa aktivointitietuepinossa edellisen aktivointitietueen tietoja. Oops!

Verkon kautta tulevat "ping"-kyselyjen tulva on niin suuri, että suoritinaikaa ei riitä muille prosesseille. Autch!

Järjestelmä ei lukkiudu tai "kaadu"

- käyttöjärjestelmän omat tietorakenteet aina (todistettavasti) eheitä
- sovellusohjelmat eivät voi koskea käyttöjärjestelmän tietorakenteisiin
- käyttäjät tai sovellusohjelmat eivät voi saada pääkäyttäjän oikeuksia ilman lupaa

Copyright Teemu Kerola 2005

Eheyden suojaamisessa on tärkeätä pitää sovellukset erillään toisistaan ja käyttöjärjestelmästä. Erilaisten yhteiskäyttövaatimusten vuoksi tämä on useinkin hyvin vaikeata. Erityisesti verkon kautta jokaiseen verkossa olevaan koneeseen tapahtuu päivittäin useita todellisia hyökkäyksiä, joita vastaan täytyy aktiivisesti puolustautua. Tahallinen häirintä voi tapahtua tietojen väärentämisen tai suoritinajan varastamisen muodossa. Vielä nykyäänkin käyttöjärjestelmissä on vielä myös sellaisia puutteita, että virheellinen ohjelma voi myös vahingossa sotkea käyttöjärjestelmän tai muiden sovellusten toimintaa.

Käyttöjärjestelmä järjestelmän eheyden turvaajana

Varauduttu kaikkii mahdollisiin virheisiin

- kaikki virheet on vaikea edes kuvitella, saati sitten varautua niihin
- vain erityisesti sallittujen toimenpiteiden suorittaminen tekee systeemistä kompelön

Sovellusohjelmat eivät voi häiritä käyttöjärjestelmää tai muita prosesseja

- tahallaan (esim. tietokonevirukset)
- vahingossa (yleisin tapaus)
- tiedon muuttaminen
- suoritinajan anastaminen

"Blue screen of death" on tyypillinen lopputulos joissain Windows-järjestelmissä systeemin kaatuessa. Siitä voi toipua vain käynnistämällä järjestelmä uudelleen.

Pääkäyttäjän salasanan tulee olla pitkä ja korkeintaan 1 kk vanha? Oletusarvoinen salasana on pakko vaihtaa hyvin pian.

Järjestelmä ei lukkiudu tai "kaadu"

- käyttöjärjestelmän omat tietorakenteet aina (todistettavasti) eheitä
- sovellusohjelmat eivät voi koskea käyttöjärjestelmän tietorakenteisiin
- käyttäjät tai sovellusohjelmat eivät voi saada pääkäyttäjän oikeuksia ilman lupaa

Copyright Teemu Kerola 2005

Väärien tietojen asemesta ehkä vielä hankalampi tilanne syntyy, jos järjestelmän tiedot eivät ole keskenään yhteneviä tai edes muodollisesti oikein. Jos esimerkiksi Windows'in käyttöjärjestelmän sisäinen tietokanta ('registry') korruptoituu, niin lääkkeeksi kelpaa helposti ainoastaan koko käyttöjärjestelmän uudelleen asennus. Tämän vuoksi käyttöjärjestelmän omien rakenteiden suojaaminen on vielä tärkeämpää kuin muiden sovellusten tietojen suojaus. Sen lisäksi, että erillisten tietoalkiciden pitää olla oikein, myös niiden muodostaman kokonaisuuden pitää olla eheä.

Käyttöjärjestelmän rakenne

Prosessien hallinta

- prosessien luonti ja tuhoaminen
- prosessien välinen viestintä
- suoritinajan jakaminen prosesseille

IPC, inter-process communication

viestit, signaalit

lukot, semaforit, tapahtumat, monitorit, postilaatikot

Muistin hallinta

- muistin määrän hallinta eri prosesseille
- prosessien muistialueiden hallinta
- yhteiskäyttö ja tiedon suojaus

Tiedostojen ja laitteiden hallinta

- tiedostojen/laitteiden lukeminen/kirjoittaminen
- yhteiskäyttö ja tiedon suojaus

Verkon hallinta

- muiden järjestelmien kanssa kommunikointi

Copyright Teemu Kerola 2005

Käyttöjärjestelmän rakenne heijastaa hyvin sen perustoimintoja. Perustoiminnat on nykyaikaiseissa käyttöjärjestelmissä toteutettu kukin omana komponenttinaan, jolloin ne on helpompi suunnitella ja sidokset muihin komponentteihin tulevat selkeästi näkyville. Ehkä tärkeimpänä komponenttina on prosessi käsitteen toteuttava prosessien hallinta. Kun prosessin käsite on käytössä, niin kaikki muut käyttöjärjestelmän palaset ovat huomattavasti helpompi rakentaa. Itse käyttöjärjestelmäkin koostuu usein useasta prosessista, jotka voivat sitten toimia itsenäisesti ja autonomisesti käyttäjäprosesseista riippumattomasti.

Käyttöjärjestelmän rakenne

Prosessien hallinta

- prosessien luonti ja tuhoaminen
- prosessien välinen viestintä
- suoritinajan jakaminen prosesseille

Muistin hallinta

- muistin määrän hallinta eri prosesseille
- prosessien muistialueiden hallinta
- yhteiskäyttö ja tiedon suojaus

Tiedostojen ja laitteiden hallinta

- tiedostojen/laitteiden lukeminen/kirjoittaminen
- yhteiskäyttö ja tiedon suojaus

Verkon hallinta

- muiden järjestelmien kanssa kommunikointi

Sama määrä muistia kaikille? Muistin määrä pysyy vakiona prosessin koko eliniän?

Onko prosessille muistista varattu alue yhtenäinen vai koostuuko se osista? Pysyykö muistista prosessille varattu alue koko ajan yhden prosessin hallussa?

Miten sallitaan muistialueen yhteiskäyttö?

Miten tiedetään etukäteen, paljonko muistia prosessi oikeasti tarvitsee?

Copyright Teemu Kerola 2005

Muistinhallintamoduuli tarkkailee jatkuvasti vapaan muistitilan määrää ja yrittää pitää sen kohtuullisena. Aina kun uusi prosessi luodaan, sille täytyy löytyä muistitilaa, tai sitten kyseinen prosessi tulee pistää jonoon odottamaan muistitilan vapautumista.

Käyttöjärjestelmän rakenne

Prosessien hallinta

- prosessien luonti ja tuhoaminen
- prosessien välinen viestintä
- suoritinajan jakaminen prosesseille

Muistin hallinta

- muistin määrän hallinta eri prosesseille
- prosessien muistialueiden hallinta
- yhteiskäyttö ja tiedon suojaus

Tiedostojen ja laitteiden hallinta

- tiedostojen/laitteiden lukeminen/kirjoittaminen
- yhteiskäyttö ja tiedon suojaus

Verkon hallinta

- muiden järjestelmien kanssa kommunikointi

Miten tiedostojärjestelmä toteutetaan?

Miten laitteiden käyttö voidaan rajata vain käyttöjärjestelmän rutiineille?

Miten varmistaa, että Liisa ei pääse käsiksi Matin tiedostoihin?
Työpaikalla tai kotona?

Copyright Teemu Kerola 2005

Tiedostoja ja laitteita hallitaan usein samalla tavalla, koska eri laitteilla käsiteltävä tieto voidaan helposti mieltää myös tiedostoksi. Tämän lähetymistavan etuna on myös suojauksen toteuttamisen yksinkertaistuminen, koska sekä tiedostoja että laitteita voidaan suojata samanlaisilla rakenteilla. Esimerkiksi Linuxissa kaikki laitteet on toteutettu tietyn tyyppisinä tiedostoina. Kun tuollaista erityistiedostoa sitten luetaan, niin järjestelmä ohjaakin luvun käytännössä sitten vaikkapa näppäimistölle.

Käyttöjärjestelmän rakenne

Prosessien hallinta

- prosessien luonti ja tuhoaminen
- prosessien välinen viestintä
- suoritinajan jakaminen prosesseille

Muistin hallinta

- muistin määrän hallinta eri prosesseille
- prosessien muistialueiden hallinta
- yhteiskäyttö ja tiedon suojaus

Tiedostojen ja laitteiden hallinta

- tiedostojen/laitteiden lukeminen/kirjoittaminen
- yhteiskäyttö ja tiedon suojaus

Verkon hallinta

- muiden järjestelmien kanssa kommunikointi

Mistä verkko-osoitteesta löytyy oman työpaikan verkkopalvelin?

Miten selain sovitetaan käyttämään läppärin langatonta verkkoa tai sen omaa kiinteää verkkkoa tilanteen mukaan?

Miten varmistaa, että Liisa ei pääse käsiksi Matin verkkopalvelimella oleviin tiedostoihin?

Copyright Teemu Kerola 2005

Verkkolaitteet ovat tietenkin oheislaitteina ihan samanlaisia kuin esimerkiksi kovalevy tai näppäimistö. Erona muihin laitteisiin on hyvin monimutkainen ja monimuotoinen verkon hallintaohjelmisto, minkä avulla verkkoa ja verkon takana olevia palveluja voidaan käyttää. Verkon ja sen palvelujen hallintaa käsitellää ihan omilla tietoliikennekursseilla, kun taas muut käyttöjärjestelmän osat käsitellään käyttöjärjestelmäkurssilla.

Käyttöjärjestelmän rakenne

Prosessien hallinta

- prosessien luonti ja tuhoaminen
- prosessien välinen viestintä
- suoritinajan jakaminen prosesseille

Muistin hallinta

- muistin määrän hallinta eri prosesseille
- prosessien muistialueiden hallinta
- yhteiskäyttö ja tiedon suojaus

Tiedostojen ja laitteiden hallinta

- tiedostojen/laitteiden lukeminen/kirjoittaminen
- yhteiskäyttö ja tiedon suojaus

Verkon hallinta

- muiden järjestelmien kanssa kommunikointi

Copyright Teemu Kerola 2005

Käyttöjärjestelmä voidaan toteuttaa hyvinkin erilaisilla lähestymistavoilla. Alkuaan useat käyttöjärjestelmät toteutettiin monoliittisina järkäleinä, joissa kaikki osat olivat sidoksissa toisiinsa ja toimivat yleensä koko ajan etuoikeutetussa tilassa. Nykyisissä järjestelmissä vain pieni käyttöjärjestelmän ydin toimii etuoikeutetussa tilassa ja pääosa käyttöjärjestelmää on erillisinä palikoina tämän ytimen päällä. Yleensä myös ainoastaan tämä ydin on millään tavalla laitteistosidonnainen, joten pääosa käyttöjärjestelmän koodia on helposti siirrettävissä laitteistosta toiseen. Esimerkiksi, prosessit ja niiden välinen kommunikointi toteutetaan ytimessä.

Käyttöjärjestelmärutiinien suorittaminen

Käyttöjärjestelmä koostuu joukosta
KJ-prosesseja ja KJ-aliohjelmia (funktioita)

- osa prosesseista ja aliohjelmista on etuoikeutettuja
- aliohjelmat voivat olla metodeja olioissa

Linux'in muistinhallintaprosessi *kswapd* pitää huolta, että vapaata muistitilaa on aina "tarpeeksi" uusia prosesseja varten.

KJ-prosessit elävät omaa elämäänsä

- esim. swapper - Linux'in muistinhallintaprosessi
- esim. laiteajurit, jos ne on toteutettu erillisinä prosesseina
- siirretään ready-to-run jonoon aina kun jokin odotettu asia tapahtuu
  - esim. kellolaitekeskeytys tai viesti tälle KJ-palvelinprosessille
- aktivoituvat vasta kun vuoronantaja antaa niille suorittimen

Windows 2000 on toteutettu suojattujen olioiden avulla. Myös prosessit ovat olioita, joihin liittyy pääsyoikeuksia.

KJ-aliohjelmat suoritetaan sillä hetkellä suorituksessa olevan prosessin ympäristössä

- aliohjelmana toteutettu käyttöjärjestelmäpalvelu aktivoituu suoran kutsun kautta
- aliohjelmana toteutettu käyttöjärjestelmäpalvelu aktivoituu keskeytyskäsittelyn kautta
  - esim. kellolaitekeskeytys tai I/O-laitekeskeytys

Copyright Teemu Kerola 2005

Käyttöjärjestelmä toteutetaan suurehkoja joukkona kj-prosesseja ja kj-aliohjelmia. Prosesseilla toteutettuja palveluja kutsutaan palvelupyyntöviestien avulla ja aliohjelmilla toteutettuja palveluja kutsutaan tavallisten aliohjelmakutsujen tai keskeytyskäsittelymekanismin avulla. Osa käyttöjärjestelmästä suoritetaan etuoikeutetussa tilassa, mikä pätee sitten sekä kj-prosesseihin että kj-aliohjelmiin. Mikä tahansa kj-rutiini voi myös kutsua muita kj-palveluja (prosesseja tai aliohjelmia) suorituksensa aikana.

Käyttöjärjestelmärutiinien suorittaminen

Käyttöjärjestelmä koostuu joukosta
KJ-prosesseja ja KJ-aliohjelmia (funktioita)

- osa prosesseista ja aliohjelmista on etuoikeutettuja
- aliohjelmat voivat olla metodeja olioissa

Linux'in muistinhallintaprosessi *kswapd* suorittaa kerran sekunnissa ja raivaa "tarpeeksi" vapaata muistitilaa varastoon.

KJ-prosessit elävät omaa elämäänsä

- esim. swapper - Linux'in muistinhallintaprosessi
- esim. laiteajurit, jos ne on toteutettu erillisinä prosesseina
- siirretään ready-to-run jonoon aina kun jokin odotettu asia tapahtuu
  - esim. kellolaitekeskeytys tai viesti tälle KJ-palvelinprosessille
- aktivoituvat vasta kun vuoronantaja antaa niille suorittimen

Windows'in *zero page* suorittaa hyvin huonolla prioriteetilla ja suoritinajan salliessa käy läpi vapaata muistitillaa ja täyttää sen nollilla.

KJ-aliohjelmat suoritetaan sillä hetkellä suorituksessa olevan prosessin ympäristössä

- aliohjelmana toteutettu käyttöjärjestelmäpalvelu aktivoituu suoran kutsun kautta
- aliohjelmana toteutettu käyttöjärjestelmäpalvelu aktivoituu keskeytyskäsittelyn kautta
  - esim. kellolaitekeskeytys tai I/O-laitekeskeytys

Copyright Teemu Kerola 2005

Kj-prosessit elävät omaa autonomista elämäänsä järjestelmässä, aivan kuten käyttäjätason prosessitkin. Erona vain on, että kj-prosessit ylläpitävät ja hallinnoivat koko järjestelmää, kun taas käyttäjätason prosessit tekevät varsinaisen tuottavan työn. KJ-prosessit aktivoituvat aina kun niitä kutsutaan viestien avulla tai kun ne itse haluavat herätä henkiin erilaisten ajastimien avulla. Tällä tavoin esimerkiksi kerran viikossa tai kerran sekunnissa saapuva siivooja on helppo toteuttaa järjestelmään.

Käyttöjärjestelmärutiinien suorittaminen

Käyttöjärjestelmä koostuu joukosta
KJ-prosesseja ja KJ-aliohjelmia (funktioita)

- osa prosesseista ja aliohjelmista on etuoikeutettuja
- aliohjelmat voivat olla metodeja olioissa

Näppäimistöltä voi ehkä lukea siten, että kutsutaan etuoikeutettua KJ-rutiinia *read_kbrd*, joka sitten toteuttaa näppäimistön odotuksen ja välittää painetun näppäimen koodin kutsujalle.

KJ-prosessit elävät omaa elämäänsä

- esim. swapper - Linux'in muistinhallintaprosessi
- esim. laiteajurit, jos ne on toteutettu erillisinä prosesseina
- siirretään ready-to-run jonoon aina kun jokin odotettu asia tapahtuu
  - esim. kellolaitekeskeytys tai viesti tälle KJ-palvelinprosessille
- aktivoituvat vasta kun vuoronantaja antaa niille suorittimen

Viestin lähetys toiselle käyttäjätason prosessille tapahtuu siten, että kutsutaan KJ-palvelua *send_msg*, joka sitten etuoikeutettuna voi toteuttaa itse viestin lähetyksen toiselle prosessille.

KJ-aliohjelmat suoritetaan sillä hetkellä suorituksessa olevan prosessin ympäristössä

- aliohjelmana toteutettu käyttöjärjestelmäpalvelu aktivoituu suoran kutsun kautta
- aliohjelmana toteutettu käyttöjärjestelmäpalvelu aktivoituu keskeytyskäsittelyn kautta
  - esim. kellolaitekeskeytys tai I/O-laitekeskeytys

Copyright Teemu Kerola 2005

Kj-aliohjelmat voidaan aktivoida myöskin joko suoraan kutsumalla niitä mistä tahansa prosessista, tai niiden oman aikataulun mukaisesti kutsumalla niitä kellolatekeskeytysrutiineista. Kummassakin tapauksessa ne suoritetaan sillä hetkellä muutenkin suorituksessa olevan prosessin ympäristössä. Tästä ei aiheudu mitään uutta tietosuojariskiä, koska käyttöjärjestelmän etuoikeutetut osat voivat joka tapauksessa päästä käsiksi mihin tahansa tietoon järjestelmässä. Käyttöjärjestelmään täytyy luottaa, ainakin tässä tapauksessa.

KJ-palveluun siirtyminen ja sieltä paluu

Tavalliset aliohjelmakutsut

- CALL → EXIT

Etuoikeutetun KJ-palvelun kutsu

- SVC → IRET

Suorituksessa olevan prosessin tekemä suora käyttöjärjestelmäpalvelun kutsu

Viestit KJ-prosesseille

- viesti → vastausviesti
- viestin lähettäjä odottaa vastausta RECEIVE-operaatiossa

Ajastimet ja muut keskeytykset

- keskeytys → keskeytyskäsittelijä IRET
- keskeytyskäsittelijä voi kutsua muita KJ-palveluja aliohjelmakutsuilla, SVC:llä tai viesteillä

Yllättävä tai jonkin ulkoisen tapahtuman aiheuttama epäsuora käyttöjärjestelmäpalvelun kutsu

Copyright Teemu Kerola 2005

Suoritusvuoro voi periaatteessa milloin tahansa siirtyä käyttöjärjestelmälle käyttäjätason prosessilta. Suoritusvuoron vaihtumista käyttöjärjestelmälle on kahta perustyyppiä. Ensinnäkin, suorituksessa oleva ohjelma voi explisiittisesti pyytää jotain tiettyä käyttöjärjestelmäpalvelua, jolloin suoritusvuoro luontevasti siirtyy tälle. Pyyntö voidaan esittää joko tavallisena aliohjelmakutsuna, SVC:nä tai sitten viestinä. Toisessa tapauksessa käyttöjärjestelmä saa suoritusvuoron keskeytyskäsittelyn kautta, jossa käyttöjärjestelmäkoodi on joko suoraan keskeytyskäsittelijässä tai sitten keskeytyskäsittelijä explisiittisesti pyytää jotain käyttöjärjestelmäpalvelua.

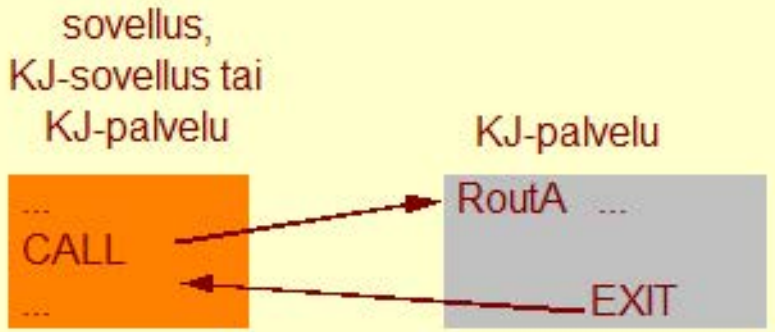
KJ-palveluun siirtyminen ja sieltä paluu

Tavalliset aliohjelmakutsut

- CALL → EXIT

Etuoikeutetun KJ-palvelun kutsu

- SVC → IRET



Viestit KJ-prosesseille

- viesti → vastausviesti
- viestin lähettäjä odottaa vastausta RECEIVE-operaatiossa

Ajastimet ja muut keskeytykset

- keskeytys → keskeytyskäsittelijä IRET
- keskeytyskäsittelijä voi kutsua muita KJ-palveluja aliohjelmakutsuilla, SVC:llä tai viesteillä

Copyright Teemu Kerola 2005

Yksinkertaisin tapa kutsua käyttöjärjestelmäpalvelua on tavallinen aliohjelmakutsu. Nykyaikaisissa käyttöjärjestelmissä usein suuri osa palveluista on toteutettu käyttäjätasoisissa palvelurutiineissa, joita kutsutaan nimenomaan tällä tavoin. Paluu kutsujan koodiin tapahtuu tavallisella aliohjelmasta paluu -käskyllä. Usea sovellus voi olla kutsunut samaa KJ-palvelua yhtäaikaa, joten, jos KJ-palvelu käyttää jotain kriittistä tietorakennetta, niin sen sisällä täytyy jollain tavalla ratkaista kyseisen tietorakenteen samanaikaisesta käytöstä aiheutuneet ongelmat.

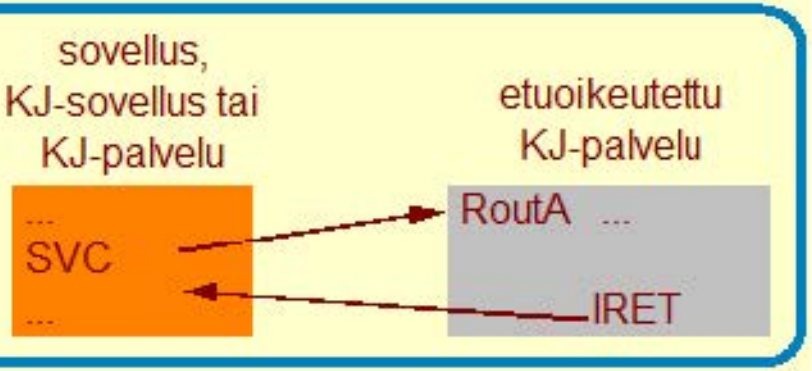
KJ-palveluun siirtyminen ja sieltä paluu

Tavalliset aliohjelmakutsut

- CALL → EXIT

Etuoikeutetun KJ-palvelun kutsu

- SVC → IRET



Viestit KJ-prosesseille

- viesti → vastausviesti
- viestin lähettäjä odottaa vastausta RECEIVE-operaatiossa

Ajastimet ja muut keskeytykset

- keskeytys → keskeytyskäsittelijä IRET
- keskeytyskäsittelijä voi kutsua muita KJ-palveluja aliohjelmakutsuilla, SVC:llä tai viesteillä

Copyright Teemu Kerola 2005

Jos kutsuttavan KJ-rutiinin halutaan suoritettavan etuoikeutetussa tilassa, niin silloin sitä pitää kutsua tavallisen aliohjelmakutsun asemesta SVC-käskyllä. Paluu kutsuvaan rutiiniin tapahtuu sitten IRET-käskyllä, joka palauttaa suorittimen tilan ennen SVC-käskyä vallinneeseen tilaan. Myös tässä tapauksessa KJ-palvelurutiinin pitää itse murehtia usean, lähes samanaikaisen kutsun aiheuttamista samanaikaisuusongelmista.

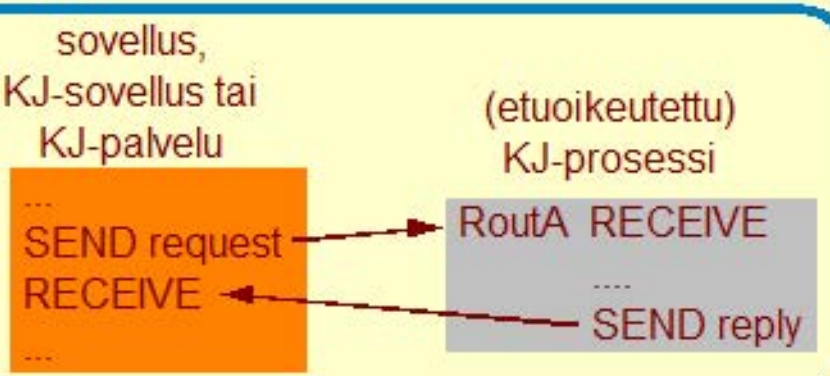
KJ-palveluun siirtyminen ja sieltä paluu

Tavalliset aliohjelmakutsut

- CALL → EXIT

Etuoikeutetun KJ-palvelun kutsu

- SVC → IRET



Viestit KJ-prosesseille

- viesti → vastausviesti
- viestin lähettäjä odottaa vastausta RECEIVE-operaatiossa

Ajastimet ja muut keskeytykset

- keskeytys → keskeytyskäsittelijä IRET
- keskeytyskäsittelijä voi kutsua muita KJ-palveluja aliohjelmakutsuilla, SVC:llä tai viesteillä

Copyright Teemu Kerola 2005

Jos käyttöjärjestelmäpalvelu on toteutettu prosessina, niin sitä ei tietenkään voi käyttää aliohjelmakutsun avulla, vaan sille pitää lähettää palvelupyyntöviesti. Palvelinprosessi saa suoritusvuoron jossain vaiheessa ja ottaa palvelupyyntöviestin vastaan ja lopulta lähettää vastausviestin kutsujalle, joka on sillä aikaa vain odottanut kyseistä vastausviestiä. Viestien lähettämisen ja vastaanoton hoitaa yleensä etuoikeutettu KJ-palvelu, jota käytetään joko CALL- tai SVC-käskyjen avulla. Palveluprosessi voi ottaa palvelupyyntöviestejä vastaan vain yhden kerrallaan, jolloin vain yksi palvelupyyntö on luontevasti kerrallaan käsittelyssä.

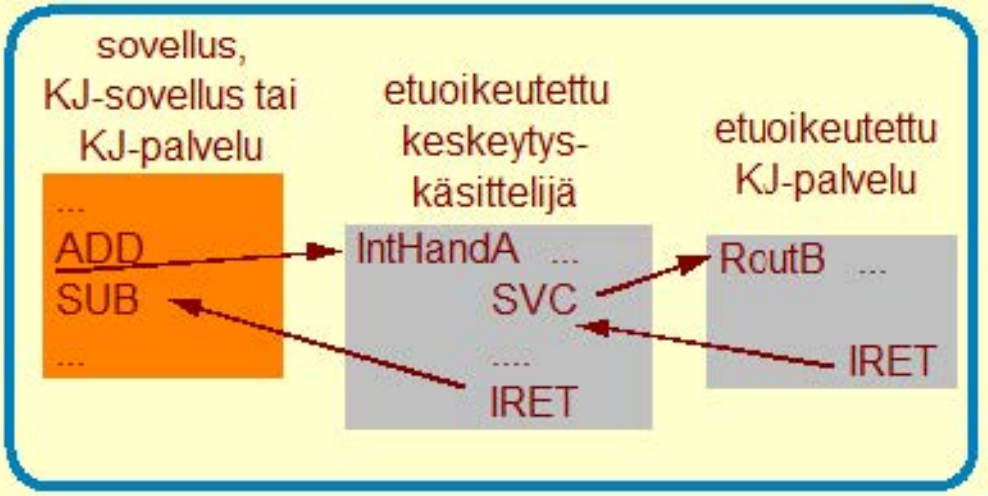
KJ-palveluun siirtyminen ja sieltä paluu

Tavalliset aliohjelmakutsut

- CALL → EXIT

Etuoikeutetun KJ-palvelun kutsu

- SVC → IRET



Viestit KJ-prosesseille

- viesti → vastausviesti
- viestin lähettäjä odottaa vastausta RECEIVE-operaatiossa

Ajastimet ja muut keskeytykset

- keskeytys → keskeytyskäsittelijä IRET
- keskeytyskäsittelijä voi kutsua muita KJ-palveluja aliohjelmakutsuilla, SVC:llä tai viesteillä

Copyright Teemu Kerola 2005

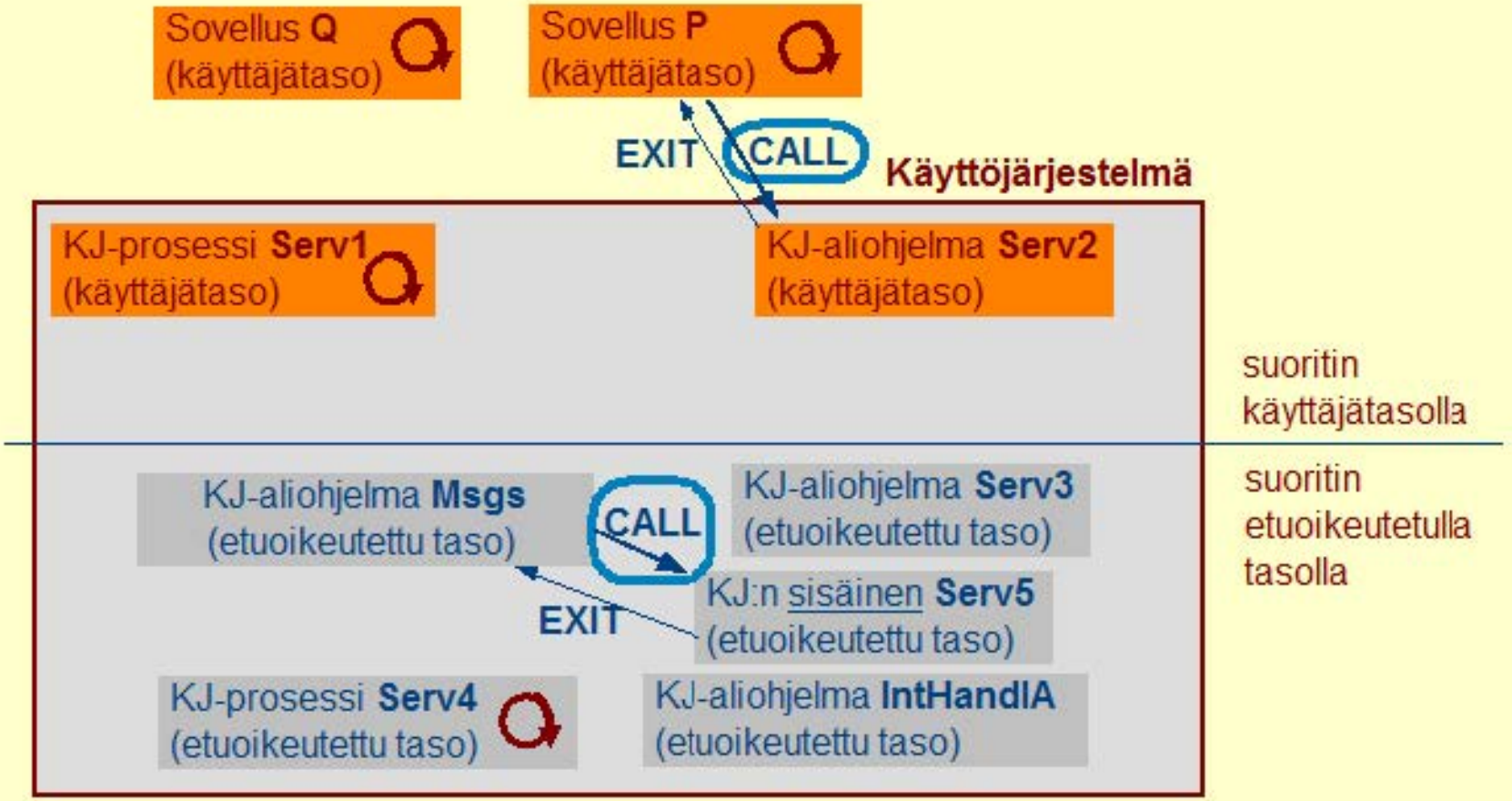
Sekä proseduraalinen että prosessilla toteutettu palvelu voi myös aktivoitua milloin tahansa keskeytyskäsittelyn kautta. Tällä tavoin toteutetaan sekä virheiden käsittely, säännölliset ajastimella toteutetut palvelut että ulkoisten I/O-laitteiden pyytämät palvelut. Tällainen keskeytys voi tapahtua milloin tahansa suorituksen aikana ja suoritusvuoron saanut käyttöjärjestelmärutiini voi myös päättää, että suoritusvuoro ei palaakaan keskeytyneelle prosessille, vaan se annetaan lopuksi jollekin toiselle, tärkeämmälle prosessille. Tämä hankaloittaa suuresti suorituksessa olevien prosessien samanaikaisuuden hallintaa.

Käyttöjärjestelmän osien suorittaminen

Copyright Teemu Kerola 2005

Tutkaillaan tässä esimerkissä tilannetta, jossa suorituksessa on kaksi normaalia sovellusprosessia (P ja Q) käyttöjärjestelmän lisäksi. Käyttöjärjestelmästä on näkyvissä sen kaksi prosessia (Serv1 ja Serv4) sekä viisi aliohjelmina toteutettua palvelua (Msgs, Serv2, Serv3, Serv5 ja IntHandlA). Prosessit toimivat itsenäisesti ilman erityisiä kutsuja ja ne on merkitty kuvaan suoritussykliä kuvaavalla symbolilla. Molemmat sovellukset ja osa käyttöjärjestelmän palveluista ovat käyttäjätasolla, mutta pääosa käyttöjärjestelmän palveluista on tässä esimerkissä etuoikeutetulla tasolla. Suoritustasolla on merkitystä palvelujen käyttäjän kannalta.

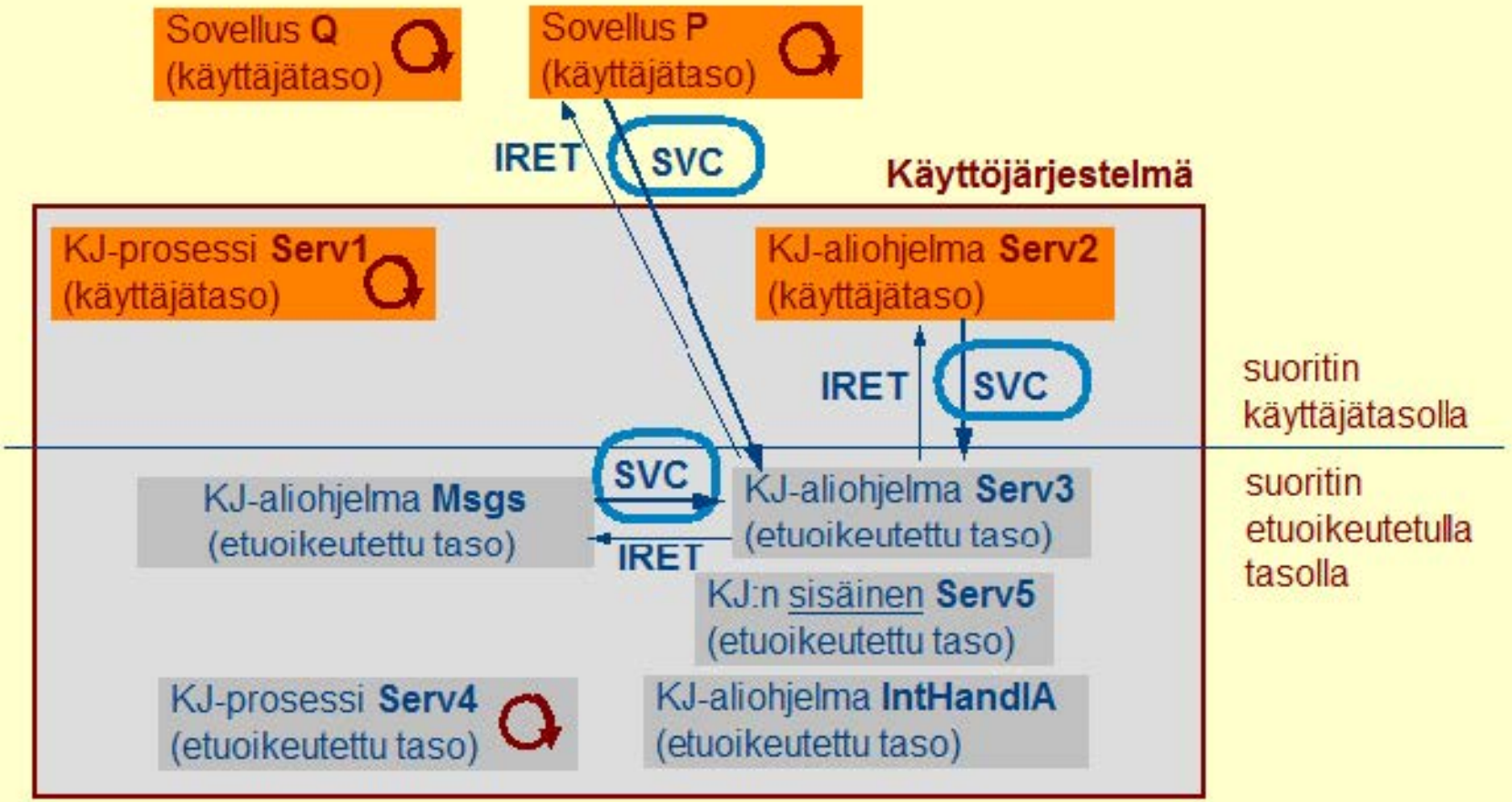
Käyttöjärjestelmän osien suorittaminen



Copyright Teemu Kerola 2005

Helpoin tapa kutsua käyttöjärjestelmää on tavallinen aliohjelmakutsu, jota kuitenkin voidaan käyttää vain jos sekä kutsuttava että kutsuttu rutiini ovat samalla suorittimen suojaustasolla. Sovellus voi siis kutsua käyttäjätasoista käyttöjärjestelmäpalvelua tai etuoikeutettu käyttöjärjestelmäpalvelu voi kutsua toista euoikeutettua käyttöjärjestelmäpalvelua. CALL-käskyllä kutsuttavasta palvelusta palataan sitten kutsuneeseen rutiiniin tavallisella aliohjelmasta paluu eli EXIT-käskyllä.

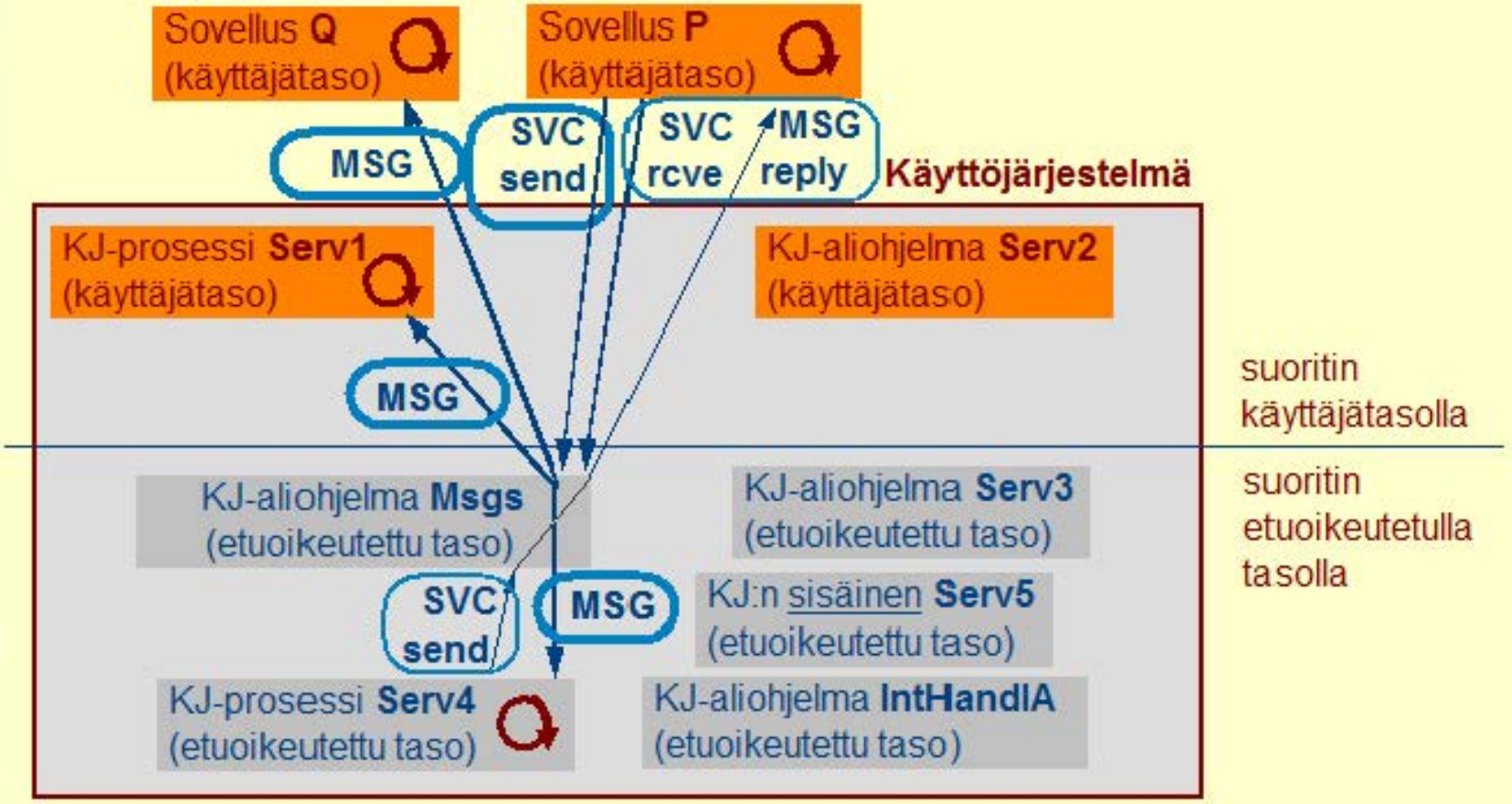
Käyttöjärjestelmän osien suorittaminen



Copyright Teemu Kerola 2005

Jos käyttöjärjestelmäpalvelu vaatii etuoikeutetun suoritustilan, niin tavallinen prosessi voi käyttää sitä vain SVC-käskyn avulla. SVC-käskyn yhteydessä suorit:imen suoritustila boostataan etuoikeutetuksi ja rutiinista palatessa IRET-käskyllä suoritustila palautuu entiselleen. Myös käyttöjärjestelmärutiinit käyttävät tällaista palvelua SVC-käskyillä, jolloin IRET-käskyn yhteydessä suorittimen suoritustila palaa ennen käskyä vallinneeseen tilaan. Sen sijaan, esimerkiksi rutiinia Serv3 ei voi kutsua CALL-käskyllä, koska Serv3:n lopussa on IRET-käsky eikä aliohjelmasta paluukäsky EXIT.

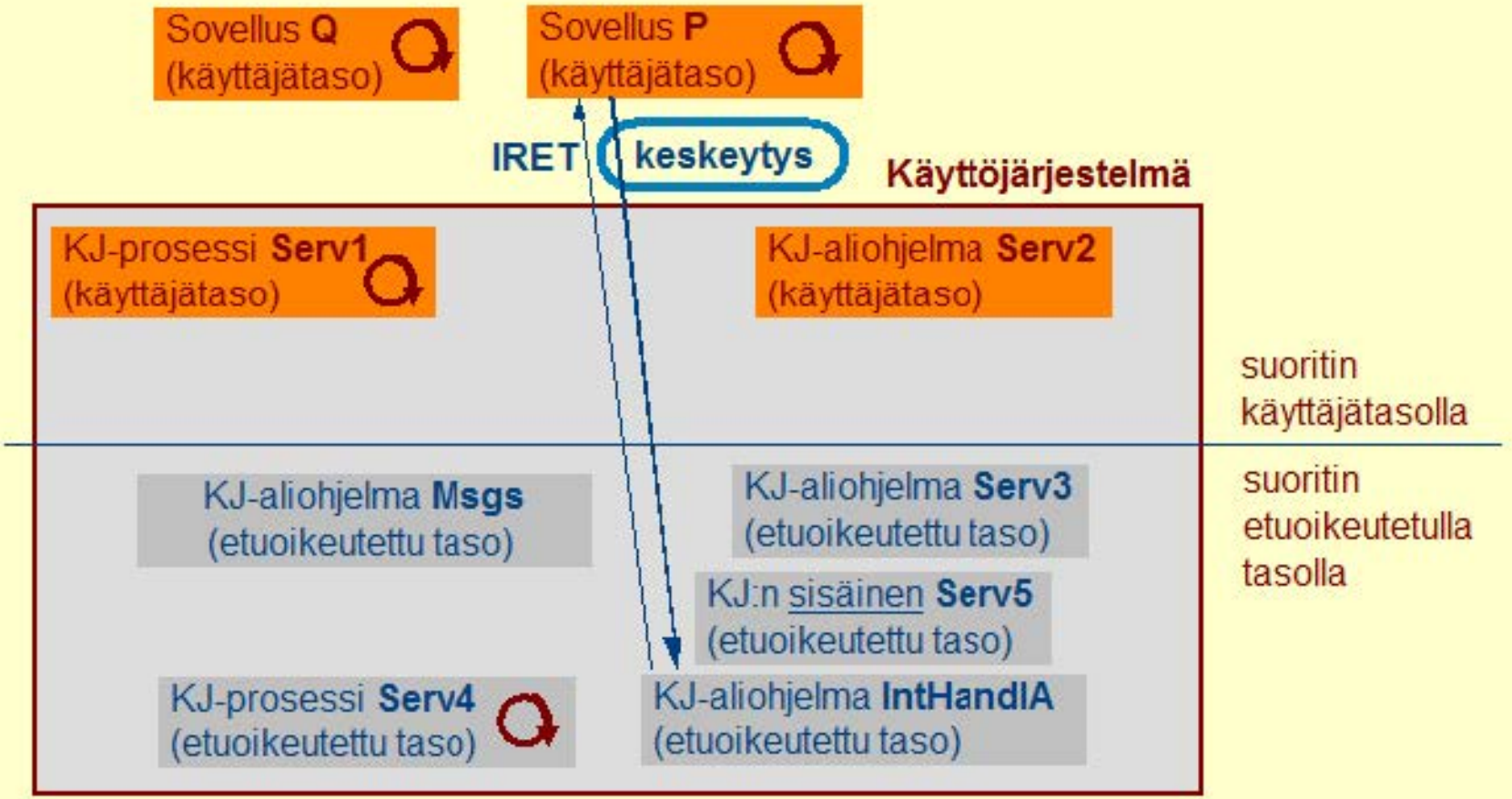
Käyttöjärjestelmän osien suorittaminen



Copyright Teemu Kerola 2005

Prosesseja ei voi kutsua suoraan CALL- tai SVC-käskyillä, mutta niille voi lähettää viestejä. Sovellustason prosessi P voi SVC-käskyn avulla pyytää viestien välityksestä huolehtivaa Msgs-palvelua lähettämään viestejä sekä muille sovelluksille että prosesseilla toteutetuille käyttöjärjestelmäpalveluille. Vastaus viestinä saatuun palvelupyyntöön annetaan sitten samalla tavalla lähetettynä vastausviestinä, jonka alkuperäinen palvelunpyytäjä ottaa vastaan. Esimerkissä vastausviestin lähetys on kuvattu prosessin Serv4 osalta. Kun prosessi P pyytää vastausviestiä SVC rcve -käskyllä, se voi joutua odotustilaan kauaksi aikaa kunnes viesti on saatavilla.

Käyttöjärjestelmän osien suorittaminen



Copyright Teemu Kerola 2005

Keskeytyskäsittelijät voivat aktivoitua milloin tahansa erilaisten poikkeustilanteiden vuoksi. Usein varsinainen työ tehdään jossain muualla kuin keskeytyskäsittelijässä, jolloin esimerkiksi laitekeskeytyksen tapahtuessa keskeytyskäsittelijä lopulta kutsuu asianmukaista laiteajuria tai lähettää sille viestin, jos se on toteutettu prosessina. Joka tapauksessa keskeytyskäsittelijä palaa lopulta IRET-käskyllä keskeytyneeseen prosessiin, jos se vaan on mahdollista. Esimerkiksi virhetilanteiden käsittelyssä on tyypillistä, että keskeytyneeseen prosessiin ei palata vaan se tapetaan ja suoritus jatkuu jostakin toisesta prosessista.

Esimerkki KJ-palvelusta: laiteajuri

Laiteajuri etuoikeutettuna aliohjelmana

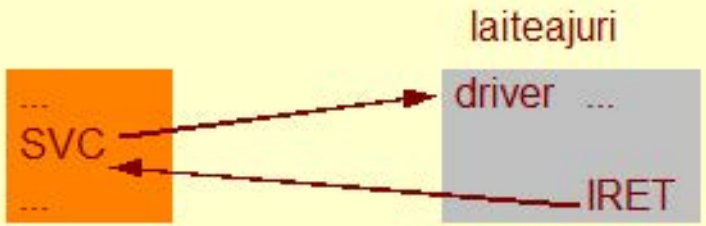
- kutsu SVC-käskyn avulla
- samanaikaisuuden hallinta laiteajurin sisällä

Laiteajuri käyttäjätilaisena aliohjelmana

- kutsu CALL-käskyn avulla
- etuoikeutettu osa toisessa SVC-käskyn avulla kutsuttavassa aliohjelmassa
- samanaikaisuuden hallinta laiteajurin sisällä

Laiteajuri prosessina

- palvelupyynnöt viestien avulla
- sovellusprosessi kutsuu laiteajurin tynkää (stub), joka sitten toteuttaa palvelun lähettämällä viestin varsinaiselle laiteajurille
- samanaikaisuuden halinta automaattista, jos vain yksi viesti kerrallaan käsittelyssä



Copyright Teemu Kerola 2005

Laiteajuri voidaan toteuttaa käyttöjärjestelmässä usealla eri tavalla. Se voidaan esimerkiksi toteuttaa yhtenä etuoikeutettuna aliohjelmana, jota kaikki sitten kutsuvat SVC-käskyn avulla. Yhtenä probleemina tässä lähestymistavassa on samanaikaisuuden hallintaongelma, mikä aiheutuu siitä, että usea eri prosessi voi käytännössä samaan aikaa kutsua tätä ajuria. Sisäisten tietorakenteiden pitäminen koherenttina vaatii ainakin aika ajoin sitä, että vain yksi palvelupyyntökutsu kerrallaan on suorituksessa ja muut ovat varmasti estettyjä. Tällaisia koodinpätkiä kutsutaan kriittisiksi vaiheiksi.

Esimerkki KJ-palvelusta: laiteajuri

Laiteajuri etuoikeutettuna aliohjelmana

- kutsu SVC-käskyn avulla
- samanaikaisuuden hallinta laiteajurin sisällä

Laiteajuri käyttäjätilaisena aliohjelmana

- kutsu CALL-käskyn avulla
- etuoikeutettu osa toisessa SVC-käskyn avulla kutsuttavassa aliohjelmassa
- samanaikaisuuden hallinta laiteajurin sisällä

Laiteajuri prosessina

- palvelupyynnöt viestien avulla
- sovellusprosessi kutsuu laiteajurin tynkää (stub), joka sitten toteuttaa palvelun lähettämällä viestin varsinaiselle laiteajurille
- samanaikaisuuden hallinta automaattista, jos vain yksi viesti kerrallaan käsittelyssä

Copyright Teemu Kerola 2005

Laiteajuri voidaan myös toteuttaa käyttäjätilaisena aliohjelmana, jolloin sitä voidaan kutsua tavallisella aliohjelmien kutsulla eli CALL-käskyllä. Osa laiteajurin toimintaa on kuitenkin laitteistosidonnaista ja vaatii välttämättä etuoikeutettua suoritustilaa. Ajuri voidaankin nyt toteuttaa siten, että vain sen välttämättömät osat ovat etuoikeutetussa tilassa ja loppuosa tavallisessa käyttäjätilassa. Tämä on myös parempi lähestymistapa suojatun käyttöjärjestelmän kannalta, missä yhtenä peruspilarina on etucikeutetun koodin mahdollisimman pieri osuus. Samanaikaisuuden hallintaongelma on edelleenkin samanlainen kuin edellisessä tapauksessa.

Esimerkki KJ-palvelusta: laiteajuri

Laiteajuri etuoikeutettuna aliohjelmana

- kutsu SVC-käskyn avulla
- samanaikaisuuden hallinta laiteajurin sisällä

Laiteajuri käyttäjätilaisena aliohjelmana

- kutsu CALL-käskyn avulla
- etuoikeutettu osa toisessa SVC-käskyn avulla kutsuttavassa aliohjelmassa
- samanaikaisuuden hallinta laiteajurin sisällä

Laiteajuri prosessina

- palvelupyynnöt viestien avulla
- sovellusprosessi kutsuu laiteajurin tynkää (stub), joka sitten toteuttaa palvelun lähettämällä viestin varsinaiselle laiteajurille
- samanaikaisuuden hallinta automaattista, jos vain yksi viesti kerrallaan käsittelyssä

Copyright Teemu Kerola 2005

Ajuri voidaan myös toteuttaa omana prosessinaan. Tällöin se näkyy sovellusprosesseille ns. tynkänä (stub), joka sitten toteuttaa varsinaisen ajurin kutsun viestin avulla. Sovellusprosessit eivät tiedä mitään ajurin toteuttamisesta prosessina, koska niille näkyy ainoastaan laiteajurin tyngän kutsurajapinta. Kun laiteajuri käsittelee vain yhden palvelupyyntöviestin kerrallaan, niin samanaikasuusongelma ei tule edes esille. Toisaalta, nykyisin laiteajuri voidaan myös toteuttaa monisäikeisenä prosessina, joissa jokaista palvelupyyntöä vastaa oma palvelusäikeensä. Samaan aikaan suorituksessa olevat säikeet saavat sitten taas jälleen aikaan samanaikaisuusongelman.

Esimerkki KJ-palvelusta: laiteajuri

```
18256 /*===========================================================================*
18257  *                                driver_task                                *
18258  *===========================================================================*/
18259 PUBLIC void driver_task(dp)
18260 struct driver *dp;      /* Device dependent entry points. */
18261 {
18262 /* Main program of any device driver task. */
18263
18264   int r, caller, proc_nr;
18265   message mess;
18266
18267   init_buffer();        /* Get a DMA buffer. */
18268
18269
18270   /* Here is the main loop of the disk task.  It waits for a message,
18271    * carries it out, and sends a reply.
18272    */
18273
18274   while (TRUE) {
18275         /* First wait for a request to read or write a disk block. */
18276         receive(ANY, &mess);
```

MINIX driver_task()

http://www.cs.vu.nl/~ast/minix.html (1.6.2005)
Copyright (c) 1987,1997, Prentice Hall, All rights reserved.

Copyright Teemu Kerola 2005

Minix'in yleisien levylaitteiden laiteajurit on toteutettu driver_task -prosesseina. Ne elävät systeemissä ikuisessa loopissa, odottaen työviestejä tiedostojen hallintaprosessilta FS_PROC_NR. Saatuaan palvelupyyntöjä, ajuri käsittelee ne yksi kerrallaan asianmukaisen rutiinin avulla, yleensä rutiinilla do_rdwr(). Lopuksi se muotoilee ja lähettää vastausviestin palvelun pyytäjälle. Rutiini do_rdwr() toteuttaa halutun laitekohtaisen operaation muotoilemalla ensin I/O-pyynnön tietueeseen ioreq ja toteuttamalla I/O-pyynnön kyseisen laitteen laitekohtaisella ajurilla dr_schedule.

Ohjelman ja käyttöjärjestelmän toteutus

- Prosessi
  - Ohjelman esitysmuoto
  - Prosessin toteutus järjestelmässä
- Käyttöjärjestelmä
  - Perustoiminnot
  - Käyttöjärjestelmäprosessit
  - Käyttöjärjestelmän toteutus
  - Kontrollin siirto prosessien välillä

Copyright Teemu Kerola 2005

Olemme nyt käyneet läpi yksittäisen ohjelman toteutuksen prosessin käsitteen avulla. Prosessi esitetään järjestelmässä yhtenä isona tietorakenteena, PCB:nä. Suoritusvuorossa olevan prosessin tietoja on myös laiterekistereissä, mutta muiden prosessien tiedot ovat ainoastaan niiden PCB:issä muistissa ja/tai levyllä. Esittelimme myös käyttöjärjestelmän perustoiminnot ja kuinka käyttöjärjestelmä voidaan toteuttaa erilaisten prosessien ja/tai palvelurutiinien avulla. Erityisesti tarkastelimme erilaisia kontrollinsiirtotapoja, joita tarvitaan käyttöjärjestelmäpalvelujen käytön yhteydessä.