LUT
University

CT30A3370 - KÄYTTÖJÄRJESTELMÄT JA SYSTEEMIOHJELMOINTI 6 OP

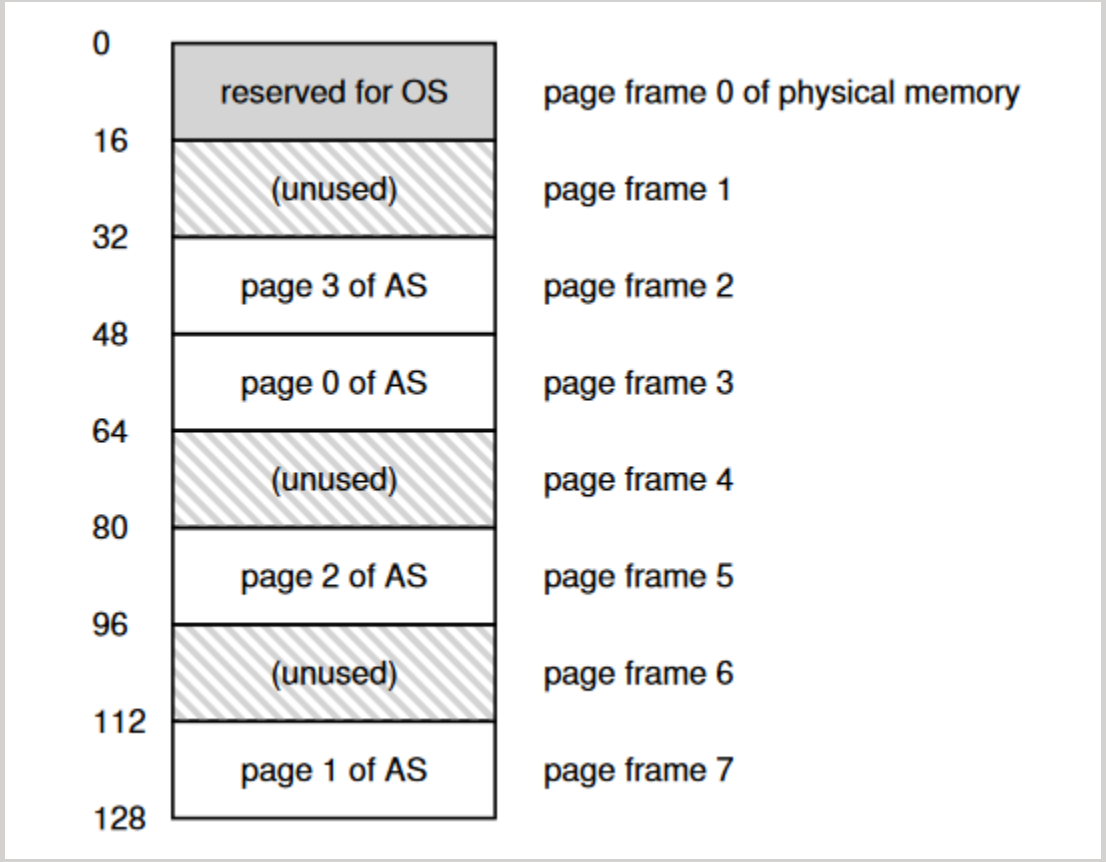
Jussi Kasurinen (etu.suku@lut.fi)

Osa kalvoista Timo Hynnisen 2016 materiaaleista

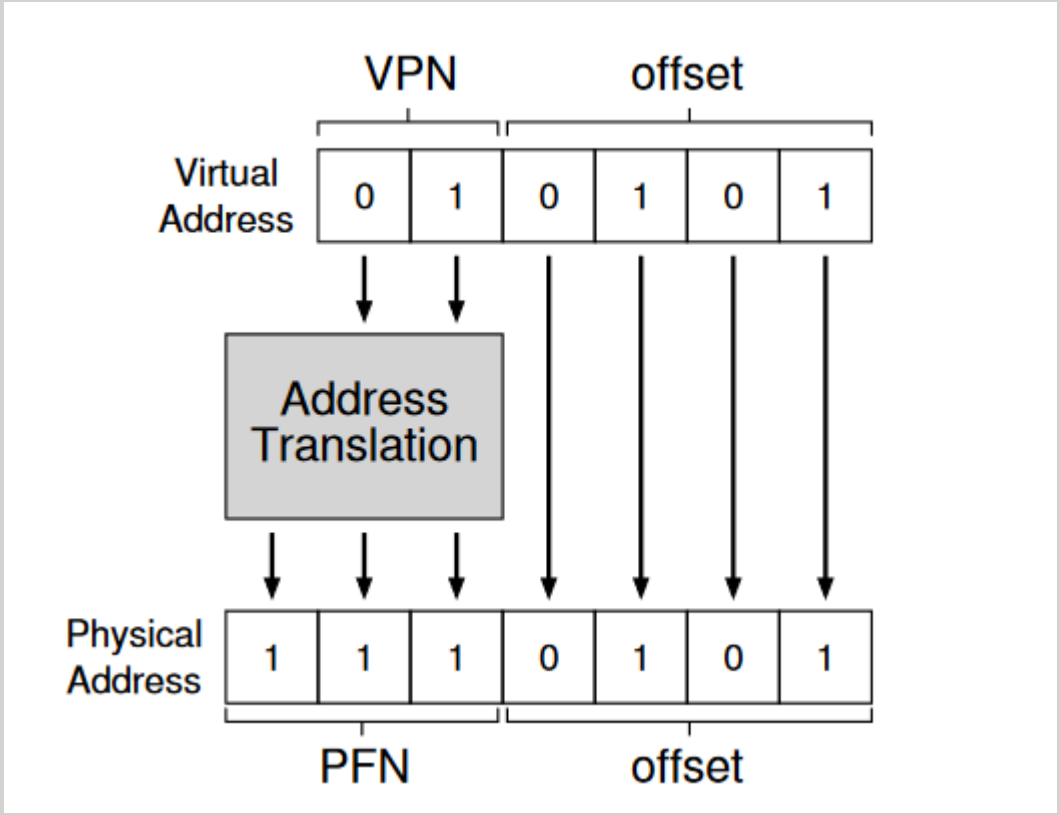
EDELLISILTÄ LUENNOILTA:

PERUSTEET MUISTISTA: SIVUTUS

- Idea: jaetaan muisti viipaleisiin.
  - Helppo pitää kirjaa mikä on käytössä ja mikä ei.
  - Helppo siirrellä paloja paikasta toiseen.
  - Helppo kirjata mitkä sivut on annettu minkäkin prosessin muistiavaruuteen.
- Käyttöjärjestelmällä on sivutustaulu (page table), jossa on tieto siitä missä mikäkin prosessin osa tai asia sijaitsee.



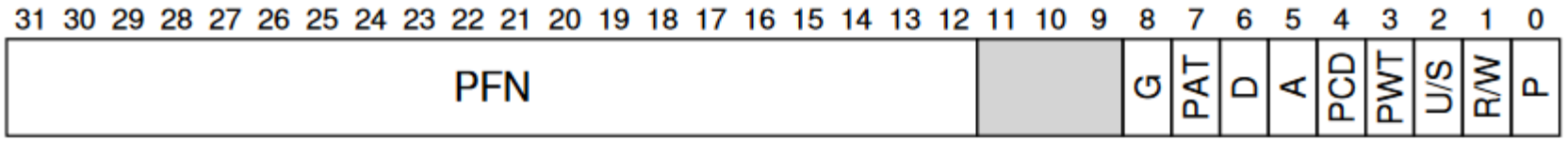
MMU JA OSOITEMUUNNOS



- Virtuaalisen osoitteen merkitsevät bitit menee muunnoksen läpi, tuloksena “oikea” osoite.
  - “Suuntanumero” vaihdetaan MMUn taulukon tietojen pohjalta.

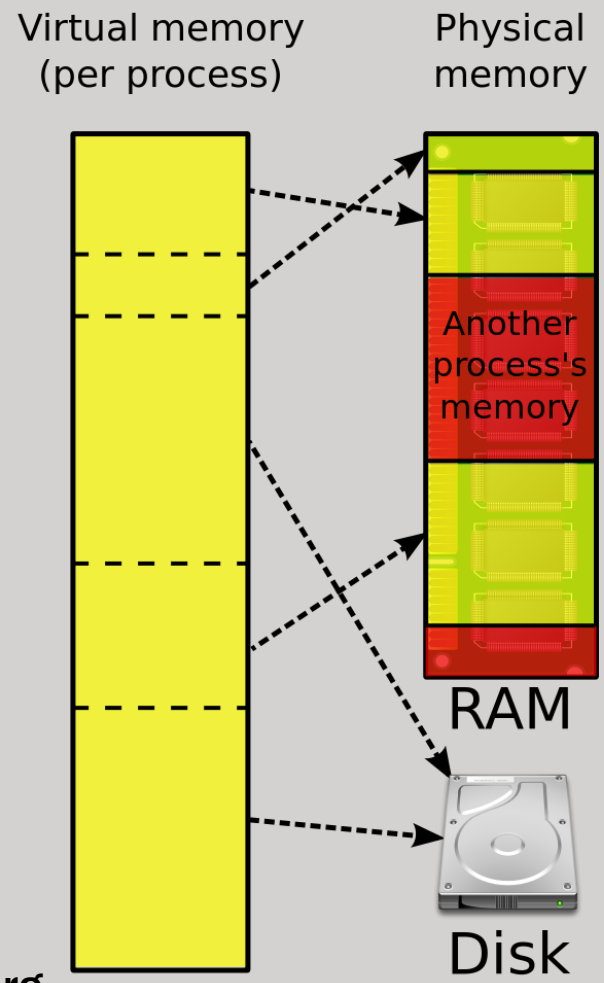
OSOITEMUUNNOSTAULUN SISÄLTÖ (X86-ESIMERKKI)

- Jos haetaan virtuaalisen sivunumeron (VPN, virtual page number) fyysistä sijaintia, mennään sivutauluun ko. sivunumeron kohdalle.
  - Tämäkin on yksinkertaistus; sivunumerointi ei välttämättä ole jono, se voi olla myös jotain muuta kuten hakemistopuu.
- Sieltä voi löytyä vaikkapa tässä formaatissa oleva tietue:



SWAPPING, HEITTOVAIHTO JA VIRTUAALIMUISTI

- Aiemmin jo mainittiin, joskus muistia pitää siirtää levylle ja sieltä pois.
  - Voidaan käsitellä muistin määrää suurempia asioita.
  - Kaiken ei tarvitse aina olla muistissa odottamassa oman prosessin ajovuoroa.
- Kokonaisuudesta puhutaan näennäismuistina, tai virtuaalimuistina.



Kuva CC BY-SA 3.0 E. Hamberg

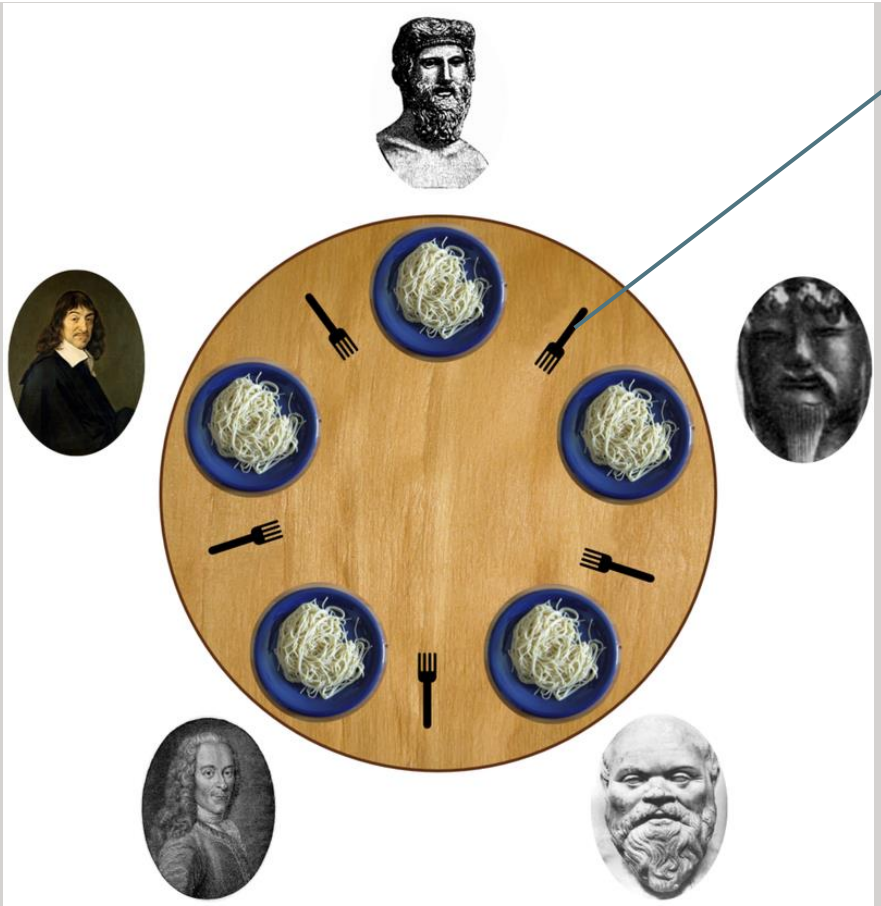
SAMANAIKAISUUS, LUKOT JA SEMAFORIT

CT30A3370 - Käyttöjärjestelmät ja systeemiohjelmointi

RE: KÄYTTÖJÄRJESTELMÄN 3 PÄÄOMINAISUUTTA

- Käyttöjärjestelmä on siis monimutkainen, mutta samalla keskeinen osa koko tietokonetta.
- Käytännössä käyttöjärjestelmälle voidaan asettaa kolme pääominaisuutta:
  - Virtualisointi (Virtualization)
  - Samanaikaisuus (Concurrency)
  - Pysyvyys (Persistence)

RE: NÄLKÄISET FILOSOFIT



Kaikki haluaa kaksi haarukkaa; koittavat ottaa vasemmanpuoleisen. Kaikki kuolevat nälkään odottaessaan toista haarukkaa.

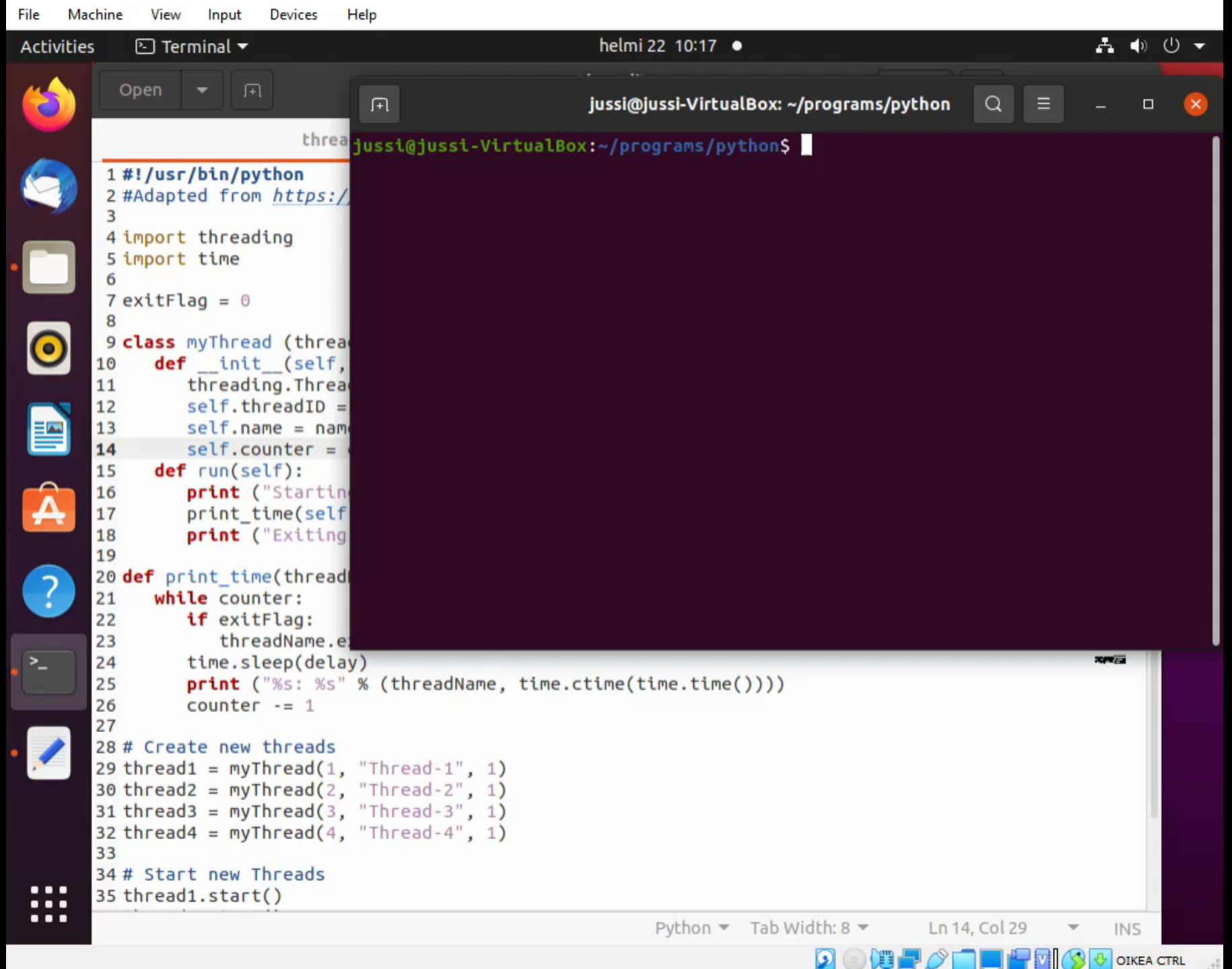
Perusongelma silloin, kun tehdään omia monisäikeisiä ohjelmia; Näitä ehkäistään atomisilla operaatioilla, lukoilla ja semaforeilla. Vuorontajaa ei nimittäin nämä ongelmat kiinnosta.

RE: SAMANAIKAISUUS

- Lyhyesti: ohjelmat saadaan toimimaan samanaikaisesti jaetuilla resursseilla.
- Monisäikeiset (Multi-thread) ohjelmat ylipäänsä saadaan toimimaan, ja että rinnakaisuus on toteuteuttu järkevästi.
- Samanaikaisuus (concurrency) = Monta asiaa yhtä aikaa yhdessä suoritinresurssissa.
- Rinnakkaisuus (parallellism) = Monta samanaikaista prosessia joita ajetaan eri suoritinresursseilla.

RE: SAMANAIKAISUUS

- Miksi tämä on sitten ongelma?


File Machine View Input Devices Help
Activities
Terminal
helmi 22 10:17
Open
jussi@jussi-VirtualBox: ~/programs/python
jussi@jussi-VirtualBox:~/programs/python$
#!/usr/bin/python
#Adapted from https:/
import threading
import time
exitFlag = 0
class myThread (threa
def __init__(self,
threading.Threa
self.threadID =
self.name = nam
self.counter =
def run(self):
print ("Startin
print_time(self
print ("Exiting
def print_time(thread
while counter:
if exitFlag:
threadName.e
time.sleep(delay)
print ("%s: %s" % (threadName, time.ctime(time.time())))
counter -= 1
# Create new threads
thread1 = myThread(1, "Thread-1", 1)
thread2 = myThread(2, "Thread-2", 1)
thread3 = myThread(3, "Thread-3", 1)
thread4 = myThread(4, "Thread-4", 1)
# Start new Threads
thread1.start()
Python
Tab Width: 8
Ln 14, Col 29
INS
OIKEA CTRL

RINNAKKAISUUS / RINNAKKAINEN SUORITUS

- Suorituksen “säie”
  - Itsenäinen Fetch / (Decode) / Execute -käskysykli.
  - Suorittimen käskysykli, mistä konekielisiä käskyjä haetaan muistista, suoritetaan käsky, siirretään käskyosoitinta eteenpäin, haetaan seuraava käsky jne..
- Yksiajo = yhden säikeen / yhden prosessin suorittamista kerralla
  - Ei voi olla rinnakkaissuorituksen kanssa mitään ongelmia, kun rinnakkaisuutta ei ole!
  - Käytännössä varmaan olisi vähän nuivaa
    - Ensimmäisissä kosketusnäytöllisissä älypuhelimissa piti valita halusiko kuunnella musiikkia vai lähetellä snäppejä teleskoopissa? Moniajo ei monissa mobiililaitteissa ensi alkuun tosiaan toiminut.
- Moniajo = monta säiettä suorituksessa samanaikaisesti
  - Historiaa: Multics, UNIX/Linux, OS/2, Windows NT/2000/XP, Mac OS X

RINNAKKAISUUS / RINNAKKAINEN SUORITUS

- Termeistä muuten sen verran: suomeksi moniajo on ihan jees termi, mutta periaatteessa pitäisi puhua **rinnakkaisesta suorituksesta**
  - Moniajo voi tarkoittaa myös muutakin
- Lähtökohta on se, että yksi suoritin haluaa tehdä monta asiaa samanaikaisesti
  - Vaikka suorittimia olisi useampikin, useimmiten tarvitsee vuorotella.

SUORITUSALUSTAN VIRTUALISOINTI

- “Suorittimelta” toiselle vaihtaminen tapahtuu tallentamalla nykyisen v-suorittimen tila ja lataamalla seuraavan muistista.
- Mutta, mikä laukaisee tämän “v-suorittimen” vaihdon?
  - Voi olla ajastin, ohjelma voi “väistää” (yield) vapaaehtoisesti, I/O (siirräntä) keskeyttää suorituksen..
  - Miten saadaan suoritus takaisin, esim. Miten saadaan suoritus takaisin vaikkapa sovellusohjelmalta takaisin käyttöjärjestelmän prosessille?
    - Tarvitaan laitteistotasolla ajastin, joka keskeyttää suorituksen.

RE: MAITOKRIISI

- Kellonajat on ulkomaanformaatissa
  - Mutta henkilökohtaisesti minusta tarina on niin tyhmä, että saattaahan tämä tapahtua myös kolmelta aamuyöstä..

	Person A	Person B
3:00	Look in fridge. Out of milk.	
3:05	Leave for store.	
3:10	Arrive at store.	Look in fridge. Out of milk.
3:15	Buy milk.	Leave for store.
3:20	Arrive home, put milk away.	Arrive at store.
3:25		Buy milk.
3:30		
		Arrive home, put milk away.
		Oh no!

MAITOKRIISI

- Muistellaan tätä maitoesimerkkiä
  - Hieno asia tietotekniikassa: helppo keksiä reaalimaailmaan sijoittuvia analogioita
  - Ainoa ero: Tietokoneet ovat tyhmiä.
- Synkronisointiongelmia voi miettiä ohjelmien moniajo-ongelmina tai ihmisten välisinä ongelmina - ihmisten pitää osata tehdä yhteistyötä

MÄÄRITELMIÄ

- Atominen operaatio
  - Suorittimen tasolla sellainen operaatio, joka ei voi keskeytyä
  - “Primitiivinen käsky”
  - Yhden konekäskyn aikana suoritin ei voi päättää keskeyttää suoritusta, vaan se on vietävä loppuun asti
    - Esimerkiksi lataus- ja tallennusoperaatiot ainakin oltava atomisia
- Synkronointi
  - Atomisten operaatioiden käyttö säikeiden yhteistoiminnan varmistamiseksi

MÄÄRITELMIÄ

- Poissulkeminen (mutual exclusion)
  - Ensimmäinen ohjelman suorittaja sulkee toisen kokonaan pois, kun itse suorittaa tehtäväänsä!
    - Toinen ohjelman suorittaja (prosessi, säie) ei pääse ollenkaan eteenpäin, sillä ensimmäinen on lukinnut niille yhteisen resurssin
- Kriittinen lohko
  - Koodilohko, jota vain yksi säie saa suorittaa kerralla.
    - Poissulkemisen tulos
    - Kriittinen lohko ja poissulkeminen ovat sama asia eri näkökantilta

MÄÄRITELMIÄ

- Lukko: estää yhtä entiteettiä tekemästä jotain
  - Lukko asetetaan paikalleen ennen kriittiseen lohkoon menemistä ja jaetun datan käsittelyä
  - Lukko poistetaan, kun tehtävä on suoritettu
  - Mitä tehdä, kun joku muu on lukinnut resurssin?
    - Odotetaan, että lukko poistuu
    - Kaikessa synkronoinnissa kaikista tärkein asia, on odottaminen
    - Tällaiset hassut lomitusongelmat, joihin törmätään, kun useampi toimija haluaa suorittaa tehtäviä rinnakkain, on kaikki ratkaistavissa odottamisella.

MÄÄRITELMIÄ

- Maito-ongelmassa homma voitaisiin ratkaista laittamalla jääkaapin ovi lukkoon
  - Todennäköisesti tämä ei kyllä ole hyvä ratkaisu? Mitä, jos kämppis haluaa jääkaapista jotain muuta (lihaa, piirakkaa, lihapiirakkaa? Muita virvoikkeita?)
  - MUTTA, synkronointiongelma tuli ratkaistua, ja ei ole liikaa maitoa!

MAITOKRIISI

```
if (noMilk) {
    if (noNote){
        leave Note;
        buy milk;
        remove note;
    }
}
```

MAITOKRIISI

- Miksi tämä ei toiminut?
  - Ihmiselle hyvä ratkaisu, tietokoneelle ei
- Muistetaan, mitä ne atomiset operaatiot oli
- Ohjelman suoritus voi keskeytyä juuri väärään aikaan
- => Liikaa maitoa, mutta vain satunnaisesti!
  - Itse asiassa vain pahentaa asiaa, koska epäonnistuu satunnaisesti!

MAITOKRIISI 2: LISÄÄ MAITOA

Säie / Prosessi 1	Säie / Prosessi 2

leave note
if (noNote){
  if (noMilk)
  buy milk
}
remove note
	leave note
	if (noNote){
	  if (noMilk)
	  buy milk
	}
	remove note

MAITOKRIISI

- Edellisessä lappu selkeästi ei lukitse jääkaappia, “resurssia”, tarpeeksi hyvin
- No mitäs jos jätetään lappu oveen ensin, ja sitten tarkastetaan tila?
- Eli ensin jätät lapun, sitten toteat että ovessa on lappu, menet pois..
  - Ei erityisen hyvä ratkaisu
- => Kukaan ei osta maitoa.

MAITOKRIISI 3: MAITOMIEHEN KOSTO

Säie / Prosessi 1	Säie / Prosessi 2

leave note A
if (noNote B){
  if (noMilk)
  buy milk
}
remove note A
	leave note B
	if (noNote A){
	  if (noMilk)
	  buy milk
	}
	remove note B

MAITOKRIISI

- Miksi on taas mahdollista, että kumpikaan ei osta maitoa?
- Konteksti vaihtuu suorittajalta toiselle juuri väärällä hetkellä, molemmat jättävät lapun vuorotellen, molemmat tekevät tarkastuksen vuorotellen ja lopuksi ottavat lapun pois vuorotellen…
- Nk. nälkiintyminen: Molemmat prosessit yrittävät varata resurssia samanaikaisesti, kumpikaan ei saa sitä käyttöönsä.
- Todella epätodennäköistä, että näin käy, mutta kun näin käy se tapahtuu juuri väärään aikaan…

MAITOKRIISI 4: MAITOHURRIKAANI

```
leave note A;
while (note B) {
    do nothing;
}
if (noMilk) {
    buy milk;
}
remove note A;
```

```
leave note B;
if (noNote A) {
    if (noMilk) {
        buy milk;
    }
}
remove note B;
```

MAITOKRIISI

- A jättää lapun A ja tarkistaa, onko B:n lappu paikalla.. Ja odottaa, kunnes lappu B häviää!
  - Lopuksi tarkistaa maitotilanteen ja hakee maitoa jos tarvis
- Näin ollen B voi jättää oman lappunsa, tarkistaa A:n tilanteen ja hakea maitoa jos tarvitsee
- Avain tässä: A:n pitää odottaa.
- Toiminta ei ole symmetristä, A ja B joutuvat toimimaan eri lailla!

MAITOKRIISI

- Toimiiko tämä?
  - Toimii, koska A ottaa huomioon kaiken, mitä B voi tehdä
    - Jos odottaa niin kauan, että toinen on jo tehnyt kaiken mahdollisen, voi huoletta alkaa suorittaa oma tehtäväänsä
  - B tietää, että jos B:n koodissa ollaan päästy A-lapun tarkastuksen ohi, niin A joko odottaa tai ei ole ollenkaan kuvioissa
- Toinen avainasia:
  - Tämä on harvinaisen pitkä ja monimutkainen ratkaisu yksinkertaiseen ongelmaan!
- Toimii, mutta epäkäytännöllistä
  - Kun tuota koodia lukee, niin eihän se nyt ole heti selvää, että tämä toimii?
- Ja mitäs, jos toimijoita (säikeitä) olisi useampia?
  - Koska ratkaisun moduulit eivät ole symmetrisiä (samanlaisia), hajoaa tämä varmasti kun jäseniä aletaan lisätä, tai jos emme tiedä monta jäseniä on.
- Kaikista pahinta: A odottaa aktiivisesti (busy-waiting), eli syö suoritinaikaa odottaessaan!
  - Odotusehdon tarkistaminen syö resursseja.

LUKITUKSEN PERIAATE

CT30A3370 - Käyttöjärjestelmät ja systeemiohjelmointi

LUKKO

- Okei, on olemassa myös parempi ratkaisu
- Me tarvitaan tietokonelaitteistolta parempia alkeisoperaatioita
- Mitä, jos meillä olisi jonkinlainen kunnollinen lukkototeutus…

```
milklock.Acquire();
        if (noMilk)
            buy milk;
milklock.Release();
```

LUKKO

- Lock.Acquire() - odottaa kunnes lukko vapautuu, ja sen jälkeen varaa resurssin
- Lock.Release() - vapauttaa lukon, ja herättää kaikki odottavat (!)
  - Eli, odottajan ei tarvitse aktiivisesti tarkistaa lukon tilaa, vaan lukko lähettää odottajalle viestin, kun se vapautuu

```
milklock.Acquire();
        if (noMilk)
            buy milk;
milklock.Release();
```

LUKKO

```
milklock.Acquire();
                if (noMilk)
                   buy milk;
milklock.Release();
```

- Lukkojen on oltava atomisia operaatioita
  - Kahden suorittajan ei ole mahdollista olla lukitsemassa samaan aikaan
- Lähes yleispätevä ratkaisu maito-ongelmaan
  - Yleispätevä, koska nyt ei enää ole mitään väliä, montako kämppäkaveria asuu saman katon alla
  - Jos useampi henkilö yrittää mennä tuon maitolukon ohi, vain yksi pääsee suorittamaan, ja muut jäävät odottamaan sitä, että ensimmäinen menee tämän tarkastusalgoritmin läpi
- Kriittinen lohko tässä on Acquiren ja Releasin välissä
  - Mutta nyt, koodin määrä on huomattavasti pienempi!

SYNKRONOINNIN ONGELMAT

- Pankkiesimerkki: Sovitaan että meillä on pankkijärjestelmä
  - Tietokoneohjelmisto joka joko…
    - Mallintaa pankkia
    - Hallinnoi pankissa ihmisten tilejä ja rahnoja
  - On asiakkaita
  - Asiakkailla on tilejä
  - Asiakas voi tallettaa tai nostaa tililtä rahaa
    - Toivottavasti vielä siten, että saldo ei mene miinukselle…

SYNKRONOINNIN ONGELMAT

- Tilille talletus: 1 säie / transaktio
  - Talletuksethan ovat toisistaan riippumattomia
  - Jos talletukset toisistaan riippumattomia => talletuksia voidaan tehdä samaan aikaan.

```
deposit(account_id, amount) {
    account = getAccount(account_id) ; /* May use disk I/O */
Account.balance += amount;
}
```

SYNKRONOINNIN ONGELMAT

- Koodi näyttää ihan kivalta, kunnes rinnakkaiset pari säiettä yrittävät tehdä samaa operaatiota samaan aikaan…
  - Kaksi asiakasta koittaa tallettaa rahaa samaan aikaan
  - Kuvitellaan vaikkapa verkkopankki, jossa muutamalle tilille koitetaan tehdä tilisiirtoa samaan aikaan
  - Toki epätodennäiköistä, että kaksi talletusta sattuisi juurikin samaan aikaan… mutta transaktioiden määrän kasvaessa yhteensattumien määrä kasvaa.

SYNKRONOINNIN ONGELMAT

- Rinnakkaiset kaksi säiettä:

`load r1, account.balance`	
	`load r1, account.balance`
	`add r1, amount`
`add r1, amount`	

SYNKRONOINNIN ONGELMAT

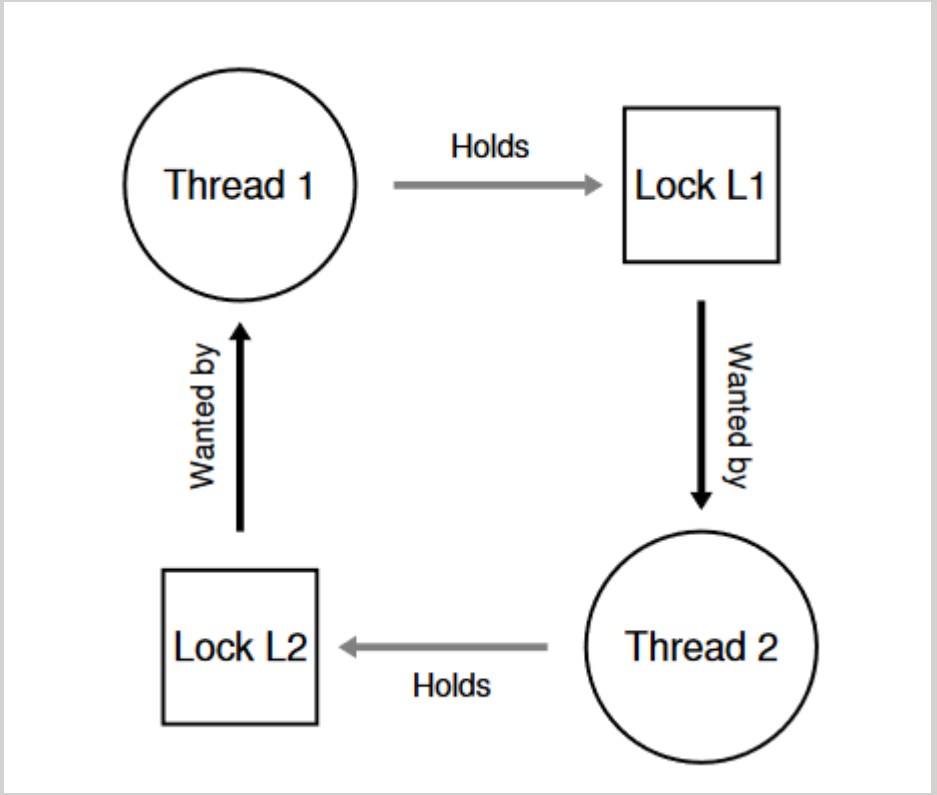
- Koska järjestelmässä on rinnakkaisuutta, lopputuloksena on odottamattomia sivuvaikutuksia
  - Järjestelmä käyttäytyy ennalta-arvaamattomasti.
  - Tässä esimerkissä rinnakkaisuus aiheuttaa sen, että toinen talletus katoaa bitttiavaruuteen.
    - Pankin toiminnan kannalta vähän nuivaa.. Tai siis, ei voi hyväksyä.
  - Itse asiassa, ainoa tapaus, jolloin talletus onnistuu on kun nämä käskyt sattuvat tapahtumaan peräkkäin.

SYNKRONOINNIN ONGELMAT

- Itse asiassa, mitäs tuossa koodissa tapahtuisikaan?
  - *Amount* on kummankin funktion oma parametri, se kyllä säilyy..
  - Mutta S2 kerkesi ladata jo rekisteriin yhden summaoperaation
    - Jos summausoperaatio tallettaa tuloksensa r1:een niin ensimmäiselle tilille lisääntyy molempien talletustapahtumien verran rahaa yhteensä?!

SYNKRONOINNIN ONGELMAT

- Toinen tavallinen ongelma on se, että systeemissä on useita lukkoja, jotka kaikki tarvitaan mutta ovat eri aikaa eri prosesseilla.
  - Varataan tietyssä järjestyksessä.
    - Esim. Linuxin kernel-dokumentaatiossa on määritelty prioriteettilista varausjärjestyksestä.
    - Käytännössä ”lukonvarauslukko”
  - ”Jos et saa kaikkia et ota yhtäkään”



ATOMISET OPERAATION

- Atominen operaatio
  - Eli operaatio, jota ei voi keskeyttää
  - Operaatio, joka suoritetaan aina kokonaan *tai* ei ollenkaan…
  - Ei voida pysäyttää kesken kaiken ja toinen konteksti ei voi muuttaa tilaa kesken kaiken
- .. ja tässä nimenomaisessa esimerkissä tavoitteena olisi jotenkin naittaa nuo kaksi riviä käskyjä, koodia, toisiinsa siten, että ne yhdessä muodostavat yhden atomisen operaation
  - Silloin tämä meidän talletustransaktio ei voi epäonnistua..
    - Kumpikin tallennus toimisi, koska saldon lataaminen ja sen korottaminen tehtäisiin aina peräkkäin.
  - Koska kahden atomisen operaation sijaan meillä olisi vain yksi

PALATAAN TAAS MAITO-ONGELMAAN

- Lukko oli täydellinen ratkaisu ™ , koska rinnakkaisuus ei enää aiheuttanut odottamattomia sivuvaikutuksia (eli ostetaan liikaa maitoa tai jätetään ostamatta kokonaan)
- Tässäkin on sama tavoite: Yritetään saada kriittinen pätkä koodia atomiseksi operaatioksi..
  - Eli, että suorituksen konteksti ei voi vahingossa vaihtua väärään aikaan.

MAITO-ONGELMA -ILTA

- Alkuperäinen ongelma (maidon ostamisen delegoimisen lisäksi..)
  - Vain muistiin tallennukset ja sieltä lataamiset olivat atomisia
  - Tästä johtuen piti tehdä tällainen monimutkainen lappusysteemi, missä aktiivisesti vahditaan, mitä toinen tekee..
- Ja kuten muistetaan, niin ratkaisu päätyi aika monimutkaiseksi.
  - .. ja kaiken lisäksi, toimii vain, kun suorittajia on kaksi
  - Useammalla suorittajalla pitäisi taas keksiä jotain muuta.

MAITO-ONGELMA -ILTA

- Huonoin asia tässä koodissa oli ehkä se, että jos A jättää lappunsa jääkaapin oveen juuri sen jälkeen, kun B on lähtenyt kaupoille, A joutuu odottamaan aktiivisesti ja tsekkaamaan tätä tilaa koko kauppareissun ajan.
  - Ja mistä tietää kauanko se kestää, voi olla tunti, viikko, vuosi…
- Jos tämän ajatuksen vie takaisin tietotekniikkaan ja suorittimeen
  - Jos A joutuu aktiivisesti odottamaa, A on koko ajan suorituksessa samalla kun B ei tee yhtään mitään!!
  - Eli kun A on suorituksessa se ei tee mitään, koska odottaa B:tä
  - … mutta jos A on suorituksessa, B ei ole
  - … eli A tekee itselleen hallaa sillä, että odottaa
    - Ei B:n suoritus nopeudu sillä, että A odottaa. B pystyy jatkamaan vain, kun A ei odota.

MAITO-ONGELMA -ILTA

- Ja totta kai on olemassa parempia tapoja tehdä vuorottelun synkronointi,
  - Mutta, tähän tarvitaan laitteistotasolla parempia primitiivejä, parempia atomisia operaatioita.
  - Parempia atomisuuksia kuin muistiin lataus ja tallennus
  - Alustalle, joka tukee parempia primitiivejä voi korkeammalla tasolla ohjelmoida monimutkaisempia mekanismeja…
- Monimutkaisemmista, korkeamman tason ratkaisuista puhutaan ainakin lukoista ja semaforeista
  - Semafori tarkoittaa, tai siis on saanut nimensä rautatien liikenneopasteesta
    - Eli siis opaste vähän niin kuin liikennevalo, saako mennä vai ei.

KESKEYTYKSET JA LUKOT

CT30A3370 - Käyttöjärjestelmät ja systeemiohjelmointi

KESKEYTYSTEN HALLINTA

- Muistellaanpa mikä oli keskeytys.
  - Mekanismi, jolla käyttis jonglööraa suoritettavia ohjelmia
  - Suorituksen siirto yhdeltä säikeeltä toiselle aika-ajoin
  - Eli kontekstin vaihdon ajoitus
  - Parhaillaan suoritettavan säikeen tila suorittimella tallennetaan keskusmuistiin, rekistereiden tila, käskyosoittimen tila (missä kohtaa ohjelmakoodia ollaan suorittamassa)…
  - Ja sitten ladataan toisen säikeen, toisen ohjelman tila suorittimelle
  - Eli näennäinen moniajo
    - Saadaan aikaan moniajoa jakamalla suoritinaikaa eri säikeiden kesken
- Voitaisiinko toteuttaa lukko keskeytysten hallinnalla?

KESKEYTYSTEN HALLINTA

- Siirryttäessä kriittiseen lohkoon koodissa, otetaan keskeytykset pois käytöstä
- Mitä oikeasti pitäisi tehdä
  - Estetään tätä laitteiston ajastinta varastamasta suoritinta silloin, kun tätä nimenomaista ohjelmaa ajetaan, koska tämä kriittinen lohko on pakko saada suoritettua!
- Mikä ongelma tässä on?
  - Ainakaan ei voida antaa käyttäjän tehdä tätä!
  - Mitä, jos parhaillaan suoritettavassa ohjelmakoodissa tapahtuu jotain…
    - Virhe
    - Ikiluuppi
    - Muuta?
  - Toinen perustavanlaatuinen vika: Mitä jos kriittinen koodilohko on yksinkertaisesti vain todella pitkä tai sen suoritus kestää kauan?
    - Yhtäkkiä koko järjestelmä odottaa yhden prosessin suorittamista ja kaikki laitteiston resurssit ovat lukossa? Ei kuulosta älykkäälle.
  - Ongelma: ei voida luottaa siihen, että säie väistää aina.

KESKEYTYSTEN HALLINTA

- Toisekseen, olette ehkä huomanneet, että tietokonejärjestelmät ovat reaaliaikaisia…
  - Mitä tarkoitan tällä? Käytännössä ei ole tilannetta, missä minkään suoritus saa estää keskeytykset
  - Jos rinnakkaisen suorituksen yhtäkkiä poistaisi, järjestelmä lakkaisi vastaamasta mihinkään syötteeseen..
    - Mitäs käyttäjä tekee jos tietokone lakkaa vastaamasta tai alkaa tökkiä? Käynnistää uudelleen..
- Entä, jos estetään vain vapaaehtoisten väistöjen aiheuttamat keskeytykset
  - Auttaisiko?
  - Moniajo toimisi kyllä, vaikka vapaaehtoisia väistöjä ei olisikaan
    - Ajastin aiheuttaisi silti ongelman

RINNAKKAISUUS KORKEAMMALLA TASOLLA

- Okei, mitä “rakennetaan parempia ja monimutkaisempia primitiivejä, joiden päälle voidaan ohjelmoida korkean tason abstrakteja mekanismeja” sitten tarkoittaa?
- No siis, tämä eri prosessien ja säikeiden vuorotus ihan suorittimen tasolla alkaa jo käydä vähän sietämättömäksi
  - Ollaan jo käyty läpi ne työkalut, millä ihan alimmalla tasolla vuoronnus tapahtuu
    - Eli osoitteenmuunnokset, virtuaalimuisti
    - Suorituksen eri tasot, eli etuoikeutettu tila ja käyttäjätila
    - Kontekstin vaihto
    - Vuoronnusalgoritmit

ATOMISET OPERAATIOT

- Ylipäätään: voidaanko toteuttaa useamman konekäskyn atomisia operaatioita?
- No, laitteistotasolla totta kai voidaan.
  - Kysehän on vain siitä, miten suoritin dekoodaa konekielisen käskyn…
- Mikä lukoissa nyt on niin erikoista, että niiden tarvitsemat mekanismit täytyy toteuttaa laitteistotasolla, mutta itse lukkoa ei voi?

ATOMISET OPERAATIOT

- No lyhyt vastaus:
  - Lukkomekanismi ei voi suoraan olla laitteistotasolla, koska silloin jokaista lukitusmekanismia kohti pitäisi käydä kernelimoodissa => kernelin läpi tapahtuvat laitteistokutsut ihan tällä tasolla olisi todella hitaita!
  - Ei siksi, etteikö laitteistopalvelut ole järkeviä toteuttaa laitteistotasolla, vaan siksi että jokaista lukkoa kohti, joita voi itse suoritettavassa sovellusohjelmassa olla vaikka kuinka ja monta, tarvitsisi tehdä monta järjestelmäkutsua... jossain vaiheessa alkaa olla hidasta
  - Toisaalta, lukkomekanismi ei voi olla täysin käyttäjätasolla...
  - Jos lukon antaa käyttäjälle suoraan, KJ menettää kontrollin lukoista => yksi prosessi voi varata muiden resurssit. Käyttäjä voi lukita koko järjestelmän...

ATOMISET OPERAATIOT

- READ - MODIFY - WRITE -käskyt
  - Erikoistuneita konekäskyjä, jotka osaavat sekä lukea muistista että kirjoittaa sinne samalla kertaa, tällä kertaa atomisesti
  - Ei kuulosta mahdottomalta?
  - Ja, tämä itse asiassa riittää lukitusmekanismien toteuttamiseen!
- Eli laitteisto vastaa “vähän monimutkaisempien mutta ei mahdottomien” primitiivien, käskyjen avulla lukituksista, mutta käskyrajapinta näkyy suoraan käyttäjälle
  - Ja käyttäjä ei pääse kuitenkaan käsiksi itse lukkomekanismiin!
  - Esimerkiksi, mitä jos jokin prosessi jättää resurssin lukkoon? Koska lukko on nyt laitteiston suojissa, käyttöjärjestelmä voi havaitessaan lukon väärinkäytön, esim. Nälkiintymisen tehdä asialle jotain!
  - Tämä siksi, että lukot eivät perustu säikeiden suoritusten keskeyttämisen (eli kontekstin vaihdon) estämiseen!
  - Kriittistä koodia ei tarvitse “saada suoritettua kerralla useita komentoja”

LUKKOJEN TOTEUTUKSEN EHDOT

- Toimivuus: ratkaisun on oikeasti estettävä muiden prosessien pääsy kriittiselle alueelle.
  - Odottaja ei sotke suorittavan prosessin asioita.
- Tasapuolisuus: kaikkien prosessien pitäisi päästä yhtälailla käyttämään lukkoa jos se on saatavilla.
  - Odottaja ei näänny.
- Tehokkuus: lukon toteutuksen (ja odottamisen) pitää olla tehokasta.
  - Odottaja ei vie resurssiaikaa.

LUKKOJEN TOTEUTUKSEN TAPOJA

- Jonolukko
- Tikettilukko
- Arpalukko
- Väistölukko (spin-lock)
- Ehtomuuttuja (semafori)

RINNAKKAISUUS KORKEAMMALLA TASOLLA

- Ja oikeastaan ei tarvitse juuri edes välittää matalan tason nippeleistä
- Mutta: Joka kerta kun vuorotuksesta on jotain puhuttu, on käsitelty vain eri säikeiden suoritusten suojaamista toisiltaan.

MITÄ TÄSTÄ LUENNOSTA PITÄÄ MUISTAA?

- Lukkojen toteutus
- Samanaikaisuuden tavallisimmat ongelmat.

HARJOITUKSET JA TIEDOTTEET

- 2. periodissa harjoitukset siirtyvät käsittelemään systeemiohjelmoinnin aiheita.
  - Apua harjoitustyöprojektien kanssa.
- Luennot jatkavat vielä 2 viikkoa levynhallinnasta, ja sen jälkeen läpikäydään ko. asioiden toteutusta oikeassa ympäristössä.

LUT
University