*Hajautetut tentti hack 1.1*

# Sanaselitykset

**Samanaikaisuus** = vaihdellaan suoritusta usean prosessin välillä, esim jos toinen odottaa syötettä (pyyntö), aletaan suorittamaan toista. Tässä tapahtuu kontekstinvaihdos (tila tallennetaan, otetaan toisen tallennettu tila, ei ilmainen operaatio)

**Rinnakkaisuus** = Useita tehtäviä voidaan suorittaa samaan aikaan useiden ytimen toimesta.

**Säie** = Prosessin sisällä olevia itsenäisesti vuoronnettavia osia. Yhteinen muistiavaruus prosessin sisällä, oma kutsupino. Käyttäjä (java… oma vuorontaja) - ja järjestelmätason säikeitä.

**Säieryhmä** = säikeet, joilla on yhteinen muistiavaruus. Ne eivät pääse muiden säieryhmien yhteiseen muistiin käsiksi. Niitä voidaan ohjata yhdessä, esim. asettaa yhteinen prioriteetti.

**Lukkiumatilanne (deadlock)** = On tilanne, jossa lukkiutumisesta seuraa, että kaksi tai useampia prosessi jumiutuu, koska ne odottavat toisiaan. Esim 1 odottaa 2 samaan aikaan kun 2 odottaa yhtä. Miten hallita? (Sivuuttaminen) Voidaan unohtaa, oletetaan että todennäköisyys tapahtua hyvin pieni tai mahdoton. (Tunnistus)Voidaan tunnistaa, ja pyrkiä avaamaan jälkeenpäin. (Ennaltaehkäisy) Varmistetaan, ettei Coffmanin ehdot täyty. Voidaan välttää numerojärjestyksellä (lukitaan aina pienempi numero ensin)

**Nälkiintyminen** = Tilanne, jossa ahne säie pitää jaettua resurssia hallussa pitkään, samalla kun muut säikeet tarvitsisivat sitä usein ja lyhkäisiä aikoja (ne nälkiintyvät). Voidaan selvittää prioriteeteilla.

**Lukko** = Mekanismi, jolla säikeiden pääsy kriittiselle (jaetun muisti) aluelle taataan yksi kerrallaan. Jos säie ottaa lukon, muilla säikeillä ei ole pääsyä alueelle. Lukon haaliminen ja vapauttaminen on atominen operaatio.

**TCP** = Yhteyksellinen tietoliikenneprotokolla. 3-vaiheinen kättely, kaksisuuntaista kommunikointia. Pakettien järjestys ja toimitus taattu.

**Soketti** = Rajapinta TCP ja UDP käyttöön sovelluksissa. Dataa voidaan lähettää tietokoneen 1 portista x tietokoneen 2 porttiin y. Kieliriippumattomia

**Protokolla** = Yhteinen sopimus siitä, missä muodossa tietoa siirretään. Binääri ja tekstipohjaisia, tilallisia ja tilattomia. Esim. http pyynnöt on muotoiltu tietyllä tavalla, molemmat osapuolet kykenevät lukemaan sitä

**Synkronoitu metodi** = tehdään metodista säieturvallinen käyttämällä oliota itsessään this-lukkona.

**Hajautettu ohjelma** koostuu useista eri järjestelmän osasta, jotka kommunikoivat keskenään, esim verkon yli. Näyttäytyy yhtenä järjestelmänä.

**Edustajaolio** (stub). Osa RMI toimintaa. Asiakkaalla on olio, joka edustaa palvelimessa olevaa etäoliota.

**Etäolio** (server object) = palvelimessa oleva olio

**Sumulaskenta** = Sumulaskenta tarkoittaa laskentatehtävien suorittamista simuloimalla ne tietokoneen avulla. Laskenta ja tiedon tallennus lähempänä loppukäyttäjiä.Etäisyys lyhyempi, nopeampi vastaus. Kaistaa säästyy (kustannukset pienenee datansirrossa), laatu ja kokemus parantuu. Kaikkea ei tarvitse siirtää loppuun asti, laskentaa voidaan tehdä jo aikaisemmin ketjussa.

**Prosessi** = Ohjelmainstanssi, joka on ladattu keskusmuistiin. Suoritin suorittaa tämän

\***UDP-protokolla** = ei takaa, että viesti päätyy perille. Yhteydetön. (osoite ja heitä paketti menemään)

**palvelutasosopimus** = asiakkaan ja palveluntarjoajan välinen sopimus, jossa määritellään palvelulle tietyt vaatimustasot

**Java Serializable rajapinta** = Voidaan lähettää olioita verkon ylitse ObjectInputStream ja ObjectOutputStream avulla.

**Säie vs prosessi:**

* Kontekstinvaihdokset nopeampia säikeissä
* Kommunikaatio nopeampaa jaetun muistin avulla.

**Miten hallita jaettuja resursseja javassa?(samanaikaisuus hallinta)**

**Thread- luokan käyttö=** Voit luoda uuden säikeen ja ajaa sen start()-metodilla. Säikeet voivat suorittaa eri töitä samanaikaisesti, parantaa ohjelman suorituskykyä.

**Synchronized** lohko vaatii olion lukoksi, myös olio itse (this). Yhdellä lukolla voidaan lukita useampia lohkoja. Monimutkaisemmat monitorilukot jättävät säikeen estyneeseen tilaan odottamaan lukon avaamista

**Volatile**, koska esim luku + kirjoitus ei ole atominen, primitiiviarvoille voidaan antaa volatile. Näin arvon päivittyessä toinen säie saa aina ajantasalla olevan arvon.

**Co-operatiivisuus wait, notify**. Tapahtuu sykronointi- lohkon sisällä. Waitin avulla vapautetaan lukko ja voidaan odottamaan estyneeseen tilaan. NotifyAll metodi ilmoittaa odottaville säikeille, että operaatiota voidaan jatkaa. Notify ilmoittaa yhdelle säikeelle.

**Kääreet ja tietorakenteet**. Takaa säieturvalliset operaatiot esim ArrayListiin. Valmiit atomiset primitiivit myös olemassa. Esim: LinkedblockingQueue, AtomicInteger, synchronizedList.

**Kehittyneet lukot.** Esim. ReentrantLock, joka sallii lukitsemisen ja avaamisen eri metodeissa. Myös ReentrantReadWriteLock mahdollistaa useiden säikeiden lukemisen samanaikaisesti, mutta vain yksi säie voi kirjoittaa kerrallaan.

**Vuoronnusmenetelmät :** tapa hallita samanaikaisuutta ja jakaa resurssit tasaisesti eri säikeille tai prosessoreille. Jaetaan:

* Co-operatiivinen (prosessi päättää itse, milloin luovuttaa prosessorin seuraavalle)
* Pre-emptiivinen (Käyttöjärjestelmä määrää CPU ajan eri prosesseille)

esim ExecutorService avulla voidaan hallita säikeiden määrää ja suorituskykyä.

**Busy waiting** = Ei pääse estyneeseen tilaan, jatkuva tarkistus täyttyykö jokin ehto)

**Demoni** (daemon thread) = taustasäie, riippuvainen muiden user threadien elinajasta. (Roskien kerääminen, taustatietojen lataaminen jne)

**3 Palvelumallia**

**SaaS** (Sovellus)

* Käyttäjät/yritykset
* Valmis ratkaisu, avaimet käteen
* Ei ylläpitoa, kehitystyötä käyttäjältä.
* Ei niin räätälöity, koska isommalle massalle
* Useita tarjoajia (

SaaS-palvelut tarjoavat valmiita ohjelmistoja asiakkaille, jotka voivat käyttää niitä verkossa maksettua kuukausimaksua vastaan. Esimerkkejä SaaS-palveluista ovat esimerkiksi sähköpostiohjelmat, projektinhallintaohjelmat ja verkkokauppasovellukset. Asiakkaat eivät tarvitse hankkia ja ylläpitää ohjelmistoja itse, vaan ne ovat saatavilla pilvipalveluna.

PaaS-palvelut tarjoavat kehittäjille alustan, jolla he voivat luoda ja julkaista verkossa toimivia sovelluksia. PaaS-palvelut tarjoavat usein myös tarvittavat työkalut ja komponentit sovellusten rakentamiseen ja ylläpitoon. Esimerkkejä PaaS-palveluista ovat esimerkiksi Amazon Web Services (AWS) Elastic Beanstalk ja Microsoft Azure App Service.

IaaS-palvelut tarjoavat asiakkaille infrastruktuuria, kuten palvelimia, tallennustilaa ja verkkoliittymiä, joita he voivat käyttää omien sovellustensa ajamiseen. Asiakkaat maksavat IaaS-palveluista käytön mukaan, ja heidän on itse huolehdittava sovellusten asentamisesta ja ylläpidosta palvelimille. Esimerkkejä IaaS-palveluista ovat esimerkiksi Amazon Web Services (AWS) EC2 ja Microsoft Azure Virtual Machines.

**PaaS** (käyttöjärjestelmät, varusohjelmat, runtimet (node))

* Kehittäjille
* Tarjoaa toimintaympäristön
* Valmis palvelu, minkä päälle rakentaa
* Google app enigne, heroku jne.
* Valittava ominaisuuksien mukaan

**IaaS** (tietokoneet, yhteydet, jonkin tason virtualisaatiota)

* Konesali
* Tallennyskapasiteettia
* OS asennettava
* ”Vapaat kädet”

**Säikeiden tilat javassa:**

****

1. Luodaan uusi säie (Thread thread = new Thread()) ja tälle käynnistetään start()-metodilla, jolloin se siirtyy ”valmis”-tilaan,.
2. Valmistilassa vuorontaja antaa sille luvan. Tästä se sitten siirtyy itse suoritus-tilaan.
3. Jos vuoronnus on pre-emptiivistä, eli vuorontaja päättää, että nyt loppuu suoritusaika, siirtyy se takaisin valmis-tilaan odottamaan lisäaikaa suoritukseen
4. Suorituksessa oleva säie voi siirtyä myös estynyt-tilaan, jos se tarvitsee mm. syötettä tai odottaa verkkopyyntöä. Kun pyyntöön on vastattu, tai uniaika loppuu, siirtyy se takaisin valmistilaan odottamaan suoritusaikaa.
5. Jos suorituksessa oleva säie saa tehtävänsä tehtyä, kuolee säie (päättynyt). Voi myös siirtyä tähän, jos heittää virheen.

**Pilvipalveluiden ominaispiirteitä:**

* Itsepalvelu (asiakas ottaa mitä tarvitsee, esim laskentateho ja käyttää tarvittavan määrän), eli resursseja tarpeen mukaan.
* Tukee eri päätelaitteet, esim Saas sovellukset, tukevat mobiili, tabletti ja työpöytä.
* resurssien yhteiskäyttö, jos joku Asiakas ei käytä resursseja, voidaan niitä jakaa muille tarvittaessa. Esim virtualisointi yhden tietokoneen päällä.
* joustava ja nopea, ei tarvitse itse hankkia lisää koneita, koodata niitä jne. Resurssien hankinta nopeaa (esim jos palveluun tulee uusia käyttäjiä.) Voidaan ostaa lisää tallennustilaa/laskentatehoa. Ei kiinteitä kuluja.
* käytön mittaaminen, laskutus (kaistan määrä, laskentateho).pyyntöjä pystytään helposti mittaamaan.
* Keskitytään liiketoimintaan, ei ylläpitoa (palvelimet ja tietoliikenne, päivitykset)

Riskit:

* Palvelun laatu? viiveet, saatavuus %.
* Kustannukset? Pitää harkita ja tehdä tarkasti…

* Tietoturva (Onko tietokoneet turvattu? Pääsy omaan dataan, luotettavuus ja tietoturvaan keskittyminen)
* Voiko muiden palvelut häiritä omaa toimintaa?
* Lainsäädäntö, pitää perehtyä
* Otetaanko varmuuskopioita?
* Riippuvaisia palveluntarjoajasta (mitä jos joskus halutaan vaihtaa pilvipalvelua?)

Edut:

1. Joustavuus. Valmiiksi tietyt proseduurit valmiina esim jos palvelinta käytetäänkin enemmän.
2. Ylläpidon helppous.
3. Voidaan keskittyä liiketoimintaan.
4. Kiinteiden kulujen minimointi.
5. IT-kulut pienenevät
6. Voidaan reagoida äkillisiin muutoksiin.

Resurssit: tallennustila, laskentakapasiteetti, tietoliikenneyhteyksiä, ylläpitoa, ohjelmia, palveluja.

Edellytykset:

* Tietotekniikan halpeneminen ja kehittyminen
* nopeat tietoliikenneyhteydet
* virtualisointi
  + piilotetaan tekniset yksityiskohdat käyttäjiltä. Abstraktio!
* resurssien hallinta
* itsepalvelullistaminen
* mittaaminen

**Tietoturva jako:**

**LES**

* Luottamuksellisuus = Ei tule paljastaa luvattomille.
* Eheys = ei voida muokata luvattomasti, tiedetään siis, että tieto on ehjää.
* Saatavuus = tulisi olla saatavilla luvallisille tahoille.

Symmetrinen (AES), asymmetrinen (RSA), asymmetrinen (DHE)

**RPC** = TCP:n tai UDP:n kautta jonkun proseduurin kutsu edustaproseduurin avulla argumentteineen, etäkutsupalvelin palauttaa tuloksen. Koska työmäärä, tai keskitetty data.

1. Ohjelma kutsuu edustaproseduuria
2. Edustaproseduuri ottaa parametrit, ja välittää viestin etäkutsupalvelimelle
3. Etäkutsupalvelin ottaa vastaan, purkaa parametrit ja tekee vaadittavat metodit
4. Tulos koodataa, ja lähetetään takaisin
5. Edustaproseduuri ottaa viestin vastaan, purkaa sen ja antaa sen ohjelmalla.

Data koodataan jollain standardiformaatilla. Esim. XDR.

**RMI** = Javan implementaatio RPC:lle. (asiakas ja palvelin javaohjelmia)

* Etäolio, Paikallinen olio, Edustajaolio**.**

RMI luokat:

* Asiakassovellus = etäkutsuihin viitataataan rajapinnan avulla, joka perii extends Remote.
* Etäolion perusteella luodaan edustajaolio
* Etäolio = Toteuttaa rajapinnan ja extends RemoteServer

**Essee aiheet:**

Essee lukoista ja miten niitä voi toteuttaa Javassa.

Essee säikeitten tiloista Javassa:

Essee pilvipalvelujen kolmesta päätyypistä:

Essee javan samanaikaisuus ongelmien ratkaisu.

**Koodi selitys:**

luokat XYS- säikeitä ja niiden roolit ja mitä tekee.

Koodiselitys, jossa lyhyt luokka (RPC) koodista ja selitä mitä muita luokkia tarvitaan.

RMI-luokka selitys ja mitä muuta tarvitaan.

XML-koodiselitystehtävä