Juuso Tuononen

Käyttöjärjestelmäarkkitehtuurien vertailu virtuaalikoneissa suoritettavien pilvipalveluiden näkökulmasta

Tietotekniikan kandidaatintutkielma

3. toukokuuta 2018

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Juuso Tuononen

Yhteystiedot: juuso.j.tuononen@student.jyu.fi

Työn nimi: Käyttöjärjestelmäarkkitehtuurien vertailu virtuaalikoneissa suoritettavien pilvipalveluiden näkökulmasta

Title in English: Comparison of operating system architectures from point of view of cloud services running on virtual machines

Työ: Kandidaatintutkielma

Sivumäärä: 21+0

Tiivistelmä: Pilvipalvelut ovat nykyään suosittuja, ja ne voidaan toteuttaa käyttäen järjestelmävirtuaalikoneita. Näissä virtuaalikoneissa käytetään usein jotakin käyttöjärjestelmää. Tämän kandidaatintutkielman tavoitteena oli selvittää mikä käyttöjärjestelmäarkkitehtuuri olisi pilvipalveluille sopivin. Tutkimusmenetelmänä käytettiin pienimuotoista kirjallisuuskatsausta. Yksiydinarkkitehtuuri vaikuttaa parhaimmalta käyttöjärjestelmäarkkitehtuurilta uusille pilvipalveluille. Lisäksi huomattiin, että käyttöjärjestelmille tarvittaisiin uusi luokittelu ja käyttöjärjestelmäarkkitehtuureita tutkittaessa tietokonelaitteiston toiminnan tunteminen on tärkeää.

Avainsanat: käyttöjärjestelmäarkkitehtuuri, mikroydin, monoliittinen ydin, pilvipalvelu, virtuaalikone, yksiydin

Abstract: Cloud services are nowadays popular and they can be implemented with system virtual machines. Some operating system is often used in system virtual machines. The objective of this bachelor thesis was to find out what operating system architecture would be the best for cloud services. A small-scale literature review was used as the research method. Unikernel architecture seems to be the best operating system architecture for new cloud services. It was also noticed that there is a need for a new operating system classification and it is important to know how the computer hardware works when doing research about operating system architectures.

Keywords: operating system architecture, microkernel, monolithic kernel, cloud service, virtual machine, unikernel

Sisältö

1 Johdanto

Pilvipalveluiden suosio on viime vuosina kasvanut ja sen myötä niiden tietoturvaan, toimintavarmuuteen ja suorituskykyyn liittyvät kysymykset ovat entistäkin tärkeämpiä. Näiden kysymysten yksi keskeinen osatekijä on pilvipalvelun ohjelmiston suorittamiseen käytettävän käyttöjärjestelmän arkkitehtuuri. Mikä käyttöjärjestelmäarkkitehtuuri olisi paras virtuaalikoneessa suoritettaville pilvipalveluille?

Nykyään myös erilaiset konttiteknologiat ovat suosittuja pilvipalveluohjelmistojen suoritusalustoja virtuaalikoneiden lisäksi. Tässä kandidaatintutkielmassa käsitellään asiaa vain virtuaalikoneiden näkökulmasta.

Tutkimusmenetelmänä tässä kandidaatintutkielmassa on pienimuotoinen kirjallisuuskatsaus. Luvussa 2 käsitellään mitä pilvipalvelu-termi oikeastaan tarkoittaa. Luvussa 3 käsitellään virtuaalikoneita. Luvussa 4 esitellään erilaisia käyttöjärjestelmäarkkitehtuureita. Luvussa 5 vertaillaan edellisessä luvussa esiteltyjä arkkitehtuureita. Luku 6 yhteenvetää kandidaatintutkielmassa käsitellyt asiat.

2 Pilvipalvelut

Tietotekniikan termitalkoot (2016) määrittelevät pilvipalvelun (engl. *cloud service*) olevan "hajautettu verkkopalvelu, jossa tietokoneita, ohjelmia, tallennustilaa ja muita tietoteknisiä palveluja käytetään verkon kautta". Lisäksi he määrittelevät pilvipalvelun synonyymiksi pilvilaskennan (engl. *cloud computing*) kanssa ja suosittelevat pilvipalvelu-termin käyttöä pilvilaskenta-termin sijasta.

Tätä tutkielmaa varten lukemassani englanninkielisessä kirjallisuudessa pilvipalvelu-termiä ei juurikaan käytetä ja pilvilaskenta-termi näyttää olevan suositumpi. Pilvipalvelu-termin määritelmä on ilmeisesti hieman erilainen englanninkielisessä kirjallisuudessa. Mell ja Grance (2011) määrittelevät National Institute of Standards and Technologyn (NIST) pilvilaskennan määritelmässä pilvilaskennan olevan malli, joka mahdollistaa verkkoyhteyden takana olevien laskentaresurssien hyödyntämisen. Esimerkkeinä laskentaresursseista määritelmässä kerrotaan tallennustila, tietoverkot, sovellusohjelmat, palvelimet ja palvelut. He määrittelevät myös kolme palvelumallia pilvilaskennalle. Näitä voi käsitykseni mukaan kutsua pilvipalvelumalleiksi. Kyseiset palvelumallit ovat Software as a Service (SaaS), Platform as a Service (PaaS) ja Infrastructure as a Service (IaaS).

Seuraavaksi esittelen edellisen kappaleen lopussa mainitut pilvipalvelumallit Mellin ja Grancen (2011) määritelmien mukaisesti. Esimerkit pilvipalvelumalleista ovat Youseffin, Butricon ja Silvan (2008) artikkelista.

- SaaS-palvelumallissa suoritetaan palveluntarjoajan sovellusohjelmia pilvi-infrastruktuurilla, eli laitteistolla joka mahdollistaa pilvilaskennan. Esimerkki tällaisesta palvelusta on Google Apps.
- PaaS-palvelumallissa asiakkaalle tarjotaan sovellusohjelmia suorittava alusta. Tällöin asiakas joutuu toteuttamaan sovellusohjelmansa kyseisen alustan rajoissa. Esimerkiksi alustaa suorittavaa käyttöjärjestelmää ei voi vaihtaa. Esimerkki tällaisesta palvelusta on Google App Engine.
- IaaS-palvelumallissa asiakkaalle tarjotaan perustavanlaatuisia laskentaresurs-

seja. Näitä ovat esimerkiksi laskenta-aika, tietoverkot ja tallennustila. Esimerkki tällaisesta palvelusta on Amazon Elastic Compute Cloud.

Youseffin, Butricon ja Silvan (2008) esittelemästä ontologiasta[1] pilvilaskennalle voi löytää edellä esitellyt pilvilaskennan palvelumallit sillä erotuksella, että IaaS-palvelumallin resurssit ovat vain laskentaan liittyviä, eli yleensä virtuaalikoneita. Tallennustilan he ovat jakaneet omaksi osa-alueeksi. Lisäksi he esittävät, että pilvilaskentaa voi ajatella kerrosarkkitehtuurina, jossa edellä esiteltyjä palvelumalleja vastaavat kerrokset eivät ole toisistaan riippuvaisia. Tällöin esimerkiksi SaaS-ohjelmisto voitaisiin toteuttaa suoraan aidon laitteiston päälle, eikä PaaS- ja IaaS-palveluiden varaan.

Pilvilaskennan käyttämisestä seuraa monia erilaisia hyötyjä. Marston ym. (2011) kertovat, että pilvilaskenta alentaa pienten yritysten kuluja ottaa liiketoiminta-analytiikka käyttöönsä ja mahdollistaa nopeamman markkinoille pääsyn monilla aloilla, koska pilvipalvelun tarvitsemaan laitteistoon ei tarvitse investoida. Lisäksi he kertovat, että pilvilaskenta mahdollistaa uudentyyppiset sovellukset ja helpottaa yritysten työtä skaalata pilvipalveluidensa käyttäjämääriä.

Tämän kandidaatintutkielman kannalta keskeisin pilvipalvelumalli on IaaS, koska virtuaalikoneita ja virtuaalikoneissa suoritettavia käyttöjärjestelmiä käytetään kyseisessä palvelumallissa. Muissa aiemmin esitellyissä palvelumalleissa ei voi itse valita käytettävää käyttöjärjestelmää.

1. Guarino, Oberle ja Staab (2009) kertovat mitä ontologia tarkoittaa tietojenkäsittelyn kontekstissa.

3 Virtuaalikoneet

Smithin ja Nairin (2005) mukaan virtuaalikoneet (engl. *virtual machine, VM*) voidaan jakaa prosessivirtuaalikoneisiin (engl. *process virtual machine*) ja järjestelmävirtuaalikoneisiin (engl. *system virtual machine*). He ovat kategorisoineet nämä kaksi kategoriaa vielä sen mukaan, suorittaako virtuaalikone isäntäkoneen (tietokone millä virtuaalikonetta suoritetaan, engl. *host*) käskykannan (engl. *instruction set architecture, ISA*) mukaista koodia. Heidän mukaansa järjestelmävirtuaalikoneita toteuttavia ohjelmistoja nimitetään virtualisointiympäristöiksi (engl. *virtual machine monitor, VMM*)[1]. Lisäksi heidän mukaansa virtuaalikoneessa suoritettavaa järjestelmää tai prosessia kutsutaan vieraaksi (engl. *guest*).

Smith ja Nair (2005) määrittelevät prosessivirtuaalikoneet virtuaalikoneiksi, jotka tarjovat sovellusohjelmille virtuaalisen sovellusohjelmointirajapinnan (engl. *application programming interface, API*) tai ohjelmabinäärirajapinnan (engl. *application binary interface, ABI*). He huomioivat isäntäkoneen käskykantaa käyttäviksi prosessivirtuaalikoneiksi nykyisten moniajokäyttöjärjestelmien prosessit ja ohjelmabinääritiedostojen dynaamiset optimoijat. Erilaista käskykantaa käyttäviä prosessivirtuaalikoneita heidän mukaansa ovat toiselle käskykannalle käännettyjä ohjelmabinääreitä suorittavat ohjelmat ja korkean tason kielten virtuaalikoneet.

Smith ja Nair (2005) määrittelevät järjestelmävirtuaalikoneen olevan virtuaalikone, joka tarjoaa monille prosesseille ja käyttöjärjestelmälle sopivan ympäristön. He kertovat isäntäkoneen käskykantaa käyttäviä järjestelmävirtuaalikoneita olevan klassiset järjestelmävirtuaalikoneet ja isännöidyt virtuaalikoneet (engl. *hosted VM*). Ero näiden kahden välillä on virtualisointiympäristön sijainti. Klassisten järjestelmävirtuaalikoneiden tapauksessa virtualisointiympäristö on toteutettu suoraan laitteiston avulla, kun taas isännöityjen virtuaalikoneiden tapauksessa virtualisointiympä-

1. Kirjallisuudessa ja WWW-sivuilla esiintyy myös englanninkielinen *hypervisor*-termi, mitä käytettään ilmeisesti synonyyminä VMM-termille. Tietotekniikan termitalkoot (2017) määrittelevät edellä mainitut termit synonyymeiksi määritelmällä "ohjelmisto tai laitteisto, joka luo ja ajaa virtuaalikoneita" ja suomennoksiksi on lueteltu sanat virtualisointiympäristö ja virtualisointialusta.

ristö on toteutettu käyttöjärjestelmän päälle. Heidän mukaan järjestelmävirtuaalikoneita, jotka eivät suorita isäntäkoneen käskykannan mukaista koodia, ovat täysijärjestelmävirtuaalikoneet (engl. *whole-system VM*) ja koodiallekirjoitetut virtuaalikoneet (engl. *codesigned VM*). Täysijärjestelmävirtuaalikoneet emuloivat sen tietokoneen prosessorin käskykantaa, mitä virtuaalikoneen on tarkoitus suorittaa. Koodiallekirjoitetut virtuaalikoneet tekevät samanlaista emulointia, kuten täysijärjestelmävirtuaalikoneet, mutta emulointi on toteutettu laitteistotasolla.

Barhamin ym. (2003) mukaan on olemassa kaksi eri näkökulmaa järjestelmävirtuaalikoneiden käyttöjärjestelmätukeen. He kirjoittavat virtuaalikoneista, mutta kyseiset virtuaalikoneet suorittavat käyttöjärjestelmää, niin voi olettaa, että kyseessä ovat järjestelmävirtuaalikoneet. Heidän mukaansa nämä ovat täysvirtualisointi (engl. *full virtualization*) ja paravirtualisointi (engl. *paravirtualization*). Täysvirtualisoinnissa käyttöjärjestelmä ei tarvitse muutoksia, että sitä voi suorittaa virtuaalikoneessa. Paravirtualisoinnissa käyttöjärjestelmä tarvitsee muutoksia ennen kuin sitä voi suorittaa virtuaalikoneessa. Heidän mukaansa paravirtualisoinnilla saavutetaan suorituskykyhyötyjä verrattuna täysvirtualisointiin.

4 Käyttöjärjestelmäarkkitehtuurit

Tässä osiossa esitellään kolme erilaista käyttöjärjestelmäarkkitehtuuria. Keskeinen erottava tekijä kaikkien kolmen välillä on käyttöjärjestelmäytimen osoiteavaruudessa suoritettavan ohjelmakoodin määrä. Mainittakoon, että kirjallisuudesta löytyy myös muitakin käyttöjärjestelmäarkkitehtuureita, kuten Englerin, Kaashoekin ja O'Toolen (1995) kehittämä eksoydinarkkitehtuuri (engl. *exokernel*).

4.1 Yksiydin

Madhavapeddy ym. (2013) kehittivät idean virtuaalikonelevykuvista, jotka ovat tarkoitettu tekemään vain yhtä asiaa, esimerkiksi suorittamaan DNS-palvelinta[1]. Nämä virtuaalikonelevykuvat luodaan kirjastokäyttöjärjestelmien (engl. *library operating system*) avulla. He nimittävät kyseisiä virtuaalikonelevykuvia yksiytimiksi (engl. *unikernel*)[2], ja ne ovat tarkoitettu pilvipalveluiden toteuttamiseen. Nimitetään tämän kandidaatintutkielman puitteissa edellä esitellyn tarkoituksen mukaisia käyttöjärjestelmiä yksiydinarkkitehtuurin omaaviksi käyttöjärjestelmiksi.

Madhavapeddyn ym. (2013) mukaan yksiydinarkkitehtuurissa kaikki ohjelmakoodi suoritetaan yhdessä osoiteavaruudessa. He mainitsevat, että mikäli ohjelmakoodi on kirjoitettu tyyppiturvallisella ohjelmointikielellä käyttöjärjestelmä voidaan suorittaa yhdessä osoiteavaruudessa. Esimerkiksi heidän enimmäkseen OCaml-ohjelmointikielellä tekemänsä Mirage-niminen käyttöjärjestelmäprototyyppi toimii kokonaan yhdessä osoiteavaruudessa.

Madhavapeddyn ym. (2013) mukaan kirjastokäyttöjärjestelmä eroaa perinteisestä käyttöjärjestelmästä siten, että perinteisen käyttöjärjestelmän palvelut, kuten vuorontaja (engl. *scheduler*) ja laiteajurit, ovat ohjelmakirjastoja, jotka linkitetään sovellusohjelman mukaan. Heidän mukaansa kirjastokäyttöjärjestelmät ja niiden toteutus moderneilla ohjelmointikielillä, jotka ovat staattisesti tyyppiturvallisia, tuottavat

1. DNS on lyhenne englanninkielisistä sanoista *Domain Name System*.
2. Suomennos on minun keksimä.

turvallisuuteen, konfigurointiin ja suorituskykyyn liittyviä hyötyjä yksiydinkäyttöjärjestelmille.

4.2 Monoliittinen ydin

Kanteen (2012) mukaan monoliittisen ytimen käyttöjärjestelmäarkkitehtuurissa käyttöjärjestelmän tarjoamat palvelut sovellusohjelmille, kuten laiteajurit ja tiedostojärjestelmä, suoritetaan kaikki yhdellä etuoikeutetulla alueella (engl. *privileged domain*). Käytännössä tämä tarkoittaa ohjelmakoodin suorittamista yhdessä osoiteavaruudessa ydintilassa (engl. *kernelspace*).

Kantee (2012) kertoo kolme tyypillistä ongelmaa kyseiselle käyttöjärjestelmäarkkitehtuurille. Ensimmäisenä hän kertoo heikon kestävyyden ja turvallisuuden, koska ydintilassa suoritettavan ohjelmakoodin virheet vaikuttavat koko järjestelmään. Toisena hän kertoo rajoitukset ohjelmakoodin uudelleenkäytössä, koska esimerkiksi jotakin monoliittiseen ytimeen tarkoitettua laiteajuria ei voi suoraan käyttää muualla kuin kyseisessä ytimessä. Kolmantena hän kertoo testauksen ja kehityksen olevan monimutkaista, koska yleensä koko käyttöjärjestelmän joutuu käynnistämään uudelleen ja ohjelmakoodin virhe voi kaataa koko järjestelmän.

Nykyisin yleisesti käytössä olevissa käyttöjärjestelmissä, kuten Windows, MacOS ja Linux-ydintä käyttävät käyttöjärjestelmät, on ominaisuuksia, jotka vähentävät ydintilassa suoritettavan ohjelmakoodin määrää. Tällainen ominaisuus voi olla vaikkapa käyttäjätilassa suoritettavat USB-laiteajurit tai tiedostojärjestelmät. Täten kyseisiä käyttöjärjestelmiä ei varmaankaan voi suoraan sanoa monoliittisen ytimen käyttöjärjestelmäarkkitehtuurin omaaviksi käyttöjärjestelmiksi. Kirjallisuudessa on kuitenkin mielipiteitä, että kyseiset käyttöjärjestelemät ovat monoliittisia, esimerkiksi Kantee (2012) ja Dautenhahn ym. (2015), mutta asiaa pitäisi tutkia tarkemmin ennen kuin johtopäätöksiä asiasta voi tehdä.

4.3 Mikroydin

Heiserin ja Elphinstonen (2016) mukaan mikroydinkäyttöjärjestelmäarkkitehtuurissa käyttöjärjestelmäydin on minimaalinen, ja se toteuttaa tarvittavat toiminnallisuudet käyttäjätilassa suoritettaville palveluille, kuten laiteajurit ja tiedostojärjestelmä. Tällöin käyttöjärjestelmäytimen osoiteavaruudessa suoritetaan mahdollisimman vähän ohjelmakoodia.

Heiserin ja Elphinstonen (2016) mukaan prosessien välisen kommunikaation (engl. *interprocess communication, IPC*) nopeus on tärkeää mikroydinarkkitehtuurille, koska sovellusohjelmaprosessien täytyy kommunikoida käyttöjärjestelmäpalveluprosessien kanssa. Lisäksi heidän mukaansa käytössä olevan prosessorin prosessoriarkkitehtuuri vaikuttaa kontekstivaihdoksen (engl. *context switch*) nopeuteen ja on siten yksi osatekijä IPC:n nopeudessa.

Mikroydinkäyttöjärjestelmiä ei ole työpöytätietokoneissa yleisesti käytössä. Heiserin ja Elphinstonen (2016) mukaan mikroydinkäyttöjärjestelmiä on käytössä muun muassa sulautetuissa järjestelmissä (engl. *embedded systems*).

5 Käyttöjärjestelmäarkkitehtuurien vertailua

Tässä luvussa vertaillaan edellisessä luvussa käsiteltyjä käyttöjärjestelmäarkkitehtuureita pilvipalveluille oleellisista näkökulmista. Vertailu perustuu kirjallisuuskatsauksen lähteisiin ja hyviin arvauksiin.

5.1 Suorituskyky ja resurssitehokkuus

Luvussa 4.3 mainittujen kontekstinvaihdosten suorituskykyvaikutukset lienevät mahdollista minimoida kun käytetään yksiydinarkkitehtuurin käyttöjärjestelmää missä ei ole käytössä prosesseja. Tämä aiheuttanee mahdollisia suorituskykyhyötyjä kyseisille yksiydinkäyttöjärjestelmille. Toisaalta myös muutkin suorituskykyyn vaikuttavat asiat, kuten Madhavapeddyn ym. (2013) mainitsema kopioita tarvitsematon laite I/O (engl. *zero copy device input/output*), lienevät tärkeitä suorituskyvyn kannalta.

Madhavapeddy ym. (2013) osoittavat, että yksiydinarkkitehtuurilla on mahdollista päästä erittäin nopeisiin käynnistysaikoihin. He osoittavat myös, että yksiydinkäyttöjärjestelmä voi viedä vähäisesti tallennustilaa.

Käynnistysaika ja käyttöjärjestelmän käyttämän tallennustilan määrä lienee hyvin pitkälti riippuvainen käytössä olevan käyttöjärjestelmän rakenteesta. Tästä voinee tehdä sen johtopäätöksen, että mitä minimaalisempi käyttöjärjestelmä, niin sitä nopeampi käynnistysaika. Lisäksi käyttöjärjestelmä vie tällöin vähemmän tallennustilaa. Yksiydinarkkitehtuuri on vahvoilla näiden asioiden suhteen, koska kyseistä arkkitehtuuria käyttävät käyttöjärjestelmät eivät sisällä mitään ylimääräistä, koska ne ovat vain yhteen tarkoitukseen suunniteltuja.

Bratterudin ym. (2015) mukaan heidän yksiydinkäyttöjärjestelmän arkkitehtuuri mahdollistaa vähäisen muistinkäytön verrattuna Ubuntu 14.04 -käyttöjärjestelmän muistinkäyttöön. Heidän arkkitehtuurissa virtuaalikonelevykuvan muodostamisvaiheessa kaikki ylimääräinen ohjelmakoodi jätetään ohjelmabinäärin linkityksessä pois

ohjelmabinääristä. Täten muistinkäyttö lienee vähäisempää verrattuna perinteisiin käyttöjärjestelmiin myös muillakin yksiydinarkkitehtuurin omaavilla käyttöjärjestelmillä mikäli kyseisille käyttöjärjestelmille on tehty vastaavia optimointeja.

Mikroytimen kohdalla resurssitehokkuus muistin ja tallennustilan osalta lienee monoliittista ydintä parempi, koska kaikki toiminnallisuus on jaettu eri ohjelmaprosesseihin. Mikäli jotain toiminnallisuutta ei tarvitse, niin sen voi vain jättää pois virtuaalikoneen levykuvasta.

5.2 Tietoturva ja toimintavarmuus

Tanenbaumin, Herderin ja Bosin (2006) mukaan nykyisissä käyttöjärjestelmissä tietoturvan ja toimintavarmuuden saralla ongelmia aiheuttaa ohjelmakomponenttien eristämättömyys muista komponenteista ja ohjelmakoodin laajuus. Pohditaan seuraavaksi näitä kahta asiaa aiemmin esiteltyjen arkkitehtuureiden näkökulmista.

Ohjelmakomponenttien eristystä ajatellen arkkitehtuurit voidaan järjestää huonoimmasta parempaan seuraavasti: yksiydin, monoliittinen ydin ja mikroydin. Tämä selittyy sillä, että ohjelmakoodin suoritus samassa osoiteavaruudessa vähenee mentäessä mikroydinarkkitehtuuriin. Esimerkiksi mikäli yksiydinkäyttöjärjestelmässä on puskurin ylivuoto -haavoittuvuus (engl. *buffer overflow*), niin hyökkääjä voi mahdollisesti kaapata haltuunsa koko virtuaalikoneen.

Tanenbaum, Herder ja Bos (2006) mainitsevat, että mitä enemmän ohjelmassa on ohjelmakoodia, niin sitä enemmän ohjelmakoodista löytyy ohjelmointivirheitä. Ohjelmakoodin laajuus on varmaankin mahdollista pitää hallittavissa kahden näkökulman avulla. Nämä ovat käyttöjärjestelmäarkkitehtuuri ja ohjelmistoarkkitehtuuri.

Käyttöjärjestelmäarkkitehtuurilla voidaan vaikuttaa ohjelmakomponenttien laajuuteen siten, että yhdellä komponentilla pitää olla vain yksi tehtävä. Tällöin yhteen komponenttiin ei tehdä ylimääräistä toiminnallisuutta ja ohjelmakoodin määrä pysyy todennäköisesti hallittavissa. Näistä esimerkkeinä ovat yksiydin- ja mikroydinarkkitehtuuri. Yksiydinarkkitehtuurissa koko virtuaalikoneella on vain yksi teh-

tävä ja mikroydinarkkitehtuurissa käyttöjärjestelmän toiminnallisuus on jaettu eri prosesseihin.

Ohjelmistoarkkitehtuurilla voi sen sijaan vaikuttaa ohjelmakomponentin sisäiseen rakenteeseen, kuten alikomponenttien ohjelmointirajapintoihin. Mikäli ohjelmakoodi on toteutettu selkeästi, niin varmaankin ohjelmointivirheitä löytyy vähemmän.

Madhavapeddy ym. (2013) mainitsevat, että ohjelmakoodin määrää pidetään usein korreloivan ohjelmakoodin hyökkäyspinta-alan kanssa. Yksiydin- ja mikroydinarkkitehtuurissa ohjelmakoodin määrän voi minimoida helposti, joten tässä mielessä ne ovat parempia kuin monoliittisen ytimen arkkitehtuuri.

Toisaalta mikäli estetään hyökkääjän mahdollisuudet suorittaa hänen omaa ohjelmakoodia ja järjestelmässä jo olevaa ohjelmakoodia, niin tällöin nämä asiat eivät ole ongelmana. Prosessoreissa on tiettyjä ominaisuuksia, joilla voi estää ohjelmakoodin muokkauksen ja suorituksen tietyillä muistialueilla. Näillä voi estää hyökkääjän oman ohjelmakoodin suorittamisen, mikäli käytössä ei ole tulkattava ohjelmointikieli. Toinen keino on tehdä vaikeammaksi ohjelmoida sellaisia virheitä ohjelmakoodiin, jotka voivat mahdollistaa hyökkäykset. Yksi keino on käyttää tyyppi- ja muistiturvallista ohjelmointikieltä. Muistiturvallisuus estää tietyt perinteiset hyökkäysvektorit, kuten puskurin ylivuoto -haavoittuvuudet. Tämä on yksi syy miksi Madhavapeddy ym. (2013) valitsivat OCaml-ohjelmointikielen heidän yksiydinkäyttöjärjestelmänsä toteutuskieleksi.

5.3 Ohjelmistokehityksen helppous

Yksiydinarkkitehtuurin käyttöjärjestelmille ohjelmien kirjoitus eroaa perinteisestä ohjelmien kirjoittamisesta jonkin verran, koska lopullinen ohjelma on perinteisen binääritiedoston sijaan virtuaalikonelevykuva. Madhavapeddyn ym. (2013) tekemälle yksiydinkäyttöjärjestelmäprototyypille ohjelmia kehittäessä lopuksi otetaan käyttöön kirjastokäyttöjärjestelmän kirjastot. Tämä prosessi on kuitenkin automatisoitu ohjelmointityökaluissa ja käytettävien kirjastojen rajapinnat ovat samat, joten se ei ilmeisesti aiheuta lisää työtä. Toinen eroavaisuus yksiydinkäyttöjärjestelmis-

sä on Madhavapeddyn ym. (2013) mukaan se, että ohjelman konfiguraatio on liitetty ohjelman kääntämisprosessiin. Perinteisesti ohjelmat lukevat konfigurointitiedoston käynnistyessään ja asettavat asetuksensa konfigurointitiedoston mukaisesti. Konfiguroinnin siirtäminen käännösprosessiin mahdollistaa kääntäjäoptimointeja, koska tietty toiminnallisuus voidaan jättää kokonaan pois virtuaalikonelevykuvasta.

Ohjelmien ja ohjelmistokirjastojen yhteensopivuus eri järjestelmien kanssa on merkittävä asia varsinkin laajoissa ohjelmissa ohjelmistokehityksen helppoutta ajatellen. Jos ohjelmistossa on paljon lähdekoodia, niin sen uudelleenkirjoitus toiselle ohjelmointikielelle on työlästä. Yksiydinarkkitehtuurissa muiden sovellusten ja ohjelmakirjastojen käyttäminen lienee haasteellista joissain tapauksissa jos ohjelma on riippuvainen sellaisesta toiminnallisuudesta, jota ei välttämättä yksiydinkäyttöjärjestelmästä löydy. Tällainen toiminnallisuus voi olla esimerkiksi prosessien käynnistäminen.

6 Yhteenveto

Pilvipalvelusovellusten toteutuksessa voidaan käyttää järjestelmävirtuaalikoneita sovellusohjelmien ajamiseen. Järjestelmävirtuaalikoneissa käytetään jonkinlaista käyttöjärjestelmää. Tässä kandidaatintutkielmassa yritettiin selvittää, mikä käyttöjärjestelmäarkkitehtuuri olisi tarkoitukseen sopivin.

Tarkoitukseen sopivimmiksi käyttöjärjestelmäarkkitehtuureiksi valikoitui "yksiydin", "monoliittinen ydin" ja "mikroydin" -nimiset arkkitehtuurit. Yksiydinkäyttöjärjestelmä lienee suositeltavin, mikäli koko käyttöjärjestelmä ja sovellus on tehty enimmäkseen yhdellä ohjelmointikielellä, joka on tyyppi- ja muistiturvallinen. Jos käytössä oleva ohjelmointikieli ei ole tyyppi- ja muistiturvallinen, niin tällöin kannattaa valita joko mikroytimen tai monoliittisen ytimen arkkitehtuurin käyttöjärjestelmä. Tässä tapauksessa mikroydinarkkitehtuuri on tietoturvan ja toimintavarmuuden kannalta paras ja monoliittisen ytimen arkkitehtuuri on suorituskyvyn kannalta paras. Kaksi edellä mainittua arkkitehtuuria lienevät myös helpompia aiemmin tehtyjen sovellusten kannalta, koska niistä todennäköisesti löytyy tuki monille prosesseille.

Edellisessä kappaleessa esitetyt johtopäätökset ovat lähinnä suuntaa antavia. Monet käyttöjärjestelmät eivät ole ominaisuuksiltaan täysin yhteensopivia luvun 4 arkkitehtuureiden määritelmien kanssa. Lisäksi kirjallisuuskatsauksessa löytyneen kirjallisuuden perusteella vaikuttaa siltä, että monoliittisen ytimen arkkitehtuuri on nykyään yleistä tietoa, koska kirjallisuudessa ei näytetä käytettävän lähdeviitteitä siitä kirjoitettaessa. Tämä aiheuttaa sen ongelman, että kirjoittajat käyttävät jonkin verran erilaisia määritelmiä ja siitä seuraa, että käyttöjärjestelmien luokittelu tietyn käyttöjärjestelmäarkkitehtuurin omaavaksi voi erota eri kirjoittajilla.

Jatkotutkimusaiheena voisi täten olla uuden luokittelun kehittäminen käyttöjärjestelmille. Luvussa 4 mainittu ohjelmakoodin suorituksen jakaminen eri osoiteavaruuksiin voisi toimia luokittelun kehityksessä hyvänä lähtökohtana.

Kandidaatintutkielman kirjoittamista hankaloitti järkevän käyttöjärjestelmäluokit-

telun puutteen lisäksi se, ettei minulla ole vielä kokemusta laiteläheisestä ohjelmoinnista. Olisi hyvä tuntea laitteiston hyödyt ja haitat melkein täydellisesti, että pystyisi arvioimaan tarkasti eri käyttöjärjestelmäarkkitehtuureiden hyötyjä ja haittoja.

Lähteet

Barham, Paul, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt ja Andrew Warfield. 2003. "Xen and the Art of Virtualization". Teoksessa *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, 164–177. SOSP '03. Bolton Landing, NY, USA: ACM. ISBN: 1-58113-757-5. doi:10.1145/945445.945462.

Bratterud, A., A. A. Walla, H. Haugerud, P. E. Engelstad ja K. Begnum. 2015. "IncludeOS: A Minimal, Resource Efficient Unikernel for Cloud Services". Teoksessa *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, 250–257. Marraskuu. doi:10.1109/CloudCom.2015.89.

Dautenhahn, Nathan, Theodoros Kasampalis, Will Dietz, John Criswell ja Vikram Adve. 2015. "Nested Kernel: An Operating System Architecture for Intra-Kernel Privilege Separation". Teoksessa *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, 191–206. ASPLOS '15. Istanbul, Turkey: ACM. ISBN: 978-1-4503-2835-7. doi:10.1145/2694344.2694386.

Engler, D. R., M. F. Kaashoek ja J. O'Toole Jr. 1995. "Exokernel: An Operating System Architecture for Application-level Resource Management". Teoksessa *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, 251–266. SOSP '95. Copper Mountain, Colorado, USA: ACM. ISBN: 0-89791-715-4. doi:10.1145/224056.224076.

Guarino, Nicola, Daniel Oberle ja Steffen Staab. 2009. "What is an ontology?" Teoksessa *Handbook on ontologies*, toimittanut Steffen Staab ja Rudi Studer. Springer, Berlin, Heidelberg. ISBN: 978-3-540-92673-3 (electronic); 978-3-540-70999-2 (print). doi:10.1007/978-3-540-92673-3_0.

Heiser, Gernot, ja Kevin Elphinstone. 2016. "L4 Microkernels: The Lessons from 20 Years of Research and Deployment". *ACM Trans. Comput. Syst.* (New York, NY, USA) 34, numero 1 (huhtikuu): 1:1–1:29. ISSN: 0734-2071. doi:10.1145/2893177.

Kantee, Antti. 2012. *Flexible Operating System Internals: The Design and Implementation of the Anykernel and Rump Kernels*. 358. Aalto University publication series DOCTORAL DISSERTATIONS; 171/2012. Väitöskirja. Aalto University; Aalto-yliopisto. ISBN: 978-952-60-4917-5 (electronic); 978-952-60-4916-8 (printed). http://urn.fi/URN:ISBN:978-952-60-4917-5.

Madhavapeddy, Anil, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand ja Jon Crowcroft. 2013. "Unikernels: Library Operating Systems for the Cloud". Teoksessa *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, 461–472. ASPLOS '13. Houston, Texas, USA: ACM. ISBN: 978-1-4503-1870-9. doi:10.1145/2451116.2451167.

Marston, Sean, Zhi Li, Subhajyoti Bandyopadhyay, Juheng Zhang ja Anand Ghalsasi. 2011. "Cloud computing — The business perspective". *Decision Support Systems* 51 (1): 176–189. ISSN: 0167-9236. doi:10.1016/j.dss.2010.12.006.

Mell, Peter, ja Tim Grance. 2011. *The NIST definition of cloud computing.* Tekninen raportti. National Institute of Standards ja Technology. doi:10.6028/NIST.SP.800-145.

Smith, J. E., ja Ravi Nair. 2005. "The architecture of virtual machines". *Computer* 38, numero 5 (toukokuu): 32–38. ISSN: 0018-9162. doi:10.1109/MC.2005.173.

Tanenbaum, A. S., J. N. Herder ja H. Bos. 2006. "Can we make operating systems reliable and secure?" *Computer* 39, numero 5 (toukokuu): 44–51. ISSN: 0018-9162. doi:10.1109/MC.2006.156.

Tietotekniikan termitalkoot. 2016. "pilvipalvelu; etäresurssipalvelu". Viitattu 16. maaliskuuta 2018. http://www.tsk.fi/tsk/termitalkoot/fi/hakemistot-267.html?page=get_id&id=ID141&vocabulary_code=TSKTT.

———. 2017. "virtualisointiympäristö; virtualisointialusta". Viitattu 8. maaliskuuta 2018. http://www.tsk.fi/tsk/termitalkoot/hakemistot-267.html?page=get_id&id=ID492&vocabulary_code=TSKTT.

Youseff, L., M. Butrico ja D. Da Silva. 2008. “Toward a Unified Ontology of Cloud Computing”. Teoksessa *2008 Grid Computing Environments Workshop*, 1–10. Marraskuu. doi:10.1109/GCE.2008.4738443.