House-käyttöjärjestelmä

Tuomo Sipola
tusesipo@jyu.fi

8. huhtikuuta 2008

Tiivistelmä

House on Haskell-kielellä kirjoitettu käyttöjärjestelmä. Haskellilla on pystytty kuvaamaan käyttöjärjestelmän keskeinen toiminta ja viestintä laitteiston kanssa. Tässä raportissa esitellään Housen ominaisuuksia ja toteutustekniikoita. Lisäksi tarkastellaan, mitä haasteita tekijät ovat kohdanneet kirjoittaessaan käyttöjärjestelmän funktionaalisella ohjelmointikielellä.

1 Johdanto

Perinteisesti käyttöjärestelmät ja laitteistoläheiset ohjelmat on kirjoitettu C-kielellä tai assemblerilla. Shapiro kuitenkin toteaa, että korkean tason ohjelmointikielet ovat kehittyneet ja tarjoavat käyttöjärjestelmien kehittäjille uusia mahdollisuuksia. [1]

Haskellin kaltaisten kielten käyttämisestä katsotaan siis olevan käytännön hyötyä monimutkaisten ohjelmistojen kehittämisessä. Hallgren ja muut pitävät käyttöjärjestelmän toiminnan varmistamista tärkeänä asiana virheettömän toiminnan takaamiseksi. Heidän mielestään Haskellin tyyppijärjestelmä ja muistinkäytön turvallisuus helpottavat ohjelman ymmärtämistä ja tekevät siitä turvallisemman. [2]

Elphinstone ja muut väittävät, että Haskellilla kirjoitettu Osker-ydin on helpompi ymmärtää kuin matalamman tason kielellä kirjoitettu, mutta katsovat haittapuoleksi korkean tason kielille tyypilliset riippuvuudet. [3] Myös Derrinnin ja muiden mielestä funktionaaliset ohjelmointikielet tarjoavat tarvittavat keinot käyttöjärjestelmän toiminnan kuvaamiseen. [4]

Haskellilla on kirjoitettu useita käyttöjärjestelmiä. Carlier ja Bobbio kirjoittivat Haskellilla hOp-mikrokernelin C-kielisen Glasgow Haskell Compilerin (GHC) ajonaikaisen järjestelmän päälle. [5] Heidän toteutuksensa käyt-

tää karsittuja C-kirjastoja. Tässä järjestelmässä C-koodi ja Haskell ajetaan erillään. House perustuu tähän pohjatyöhön. [6]

Hallgren ja muut ovat toteuttaneet myös toisen Haskell-kernelin. Tässä Osker-kernelissä rinnakkaisuus on toteutettu Housesta poikkeavalla tavalla. He pyrkivät Oskerilla kernelin toiminnan formaaliin varmistamiseen. [2] Samankaltaisia päämääriä on Shapirolla [1] sekä Hohmuthilla ja Tewsilla [7].

Myös Kinetic-käyttöjärjestelmä on kirjoitettu Haskellilla, mutta sen perusta poikkeaa hOpista ja Housesta. [8] Hallgren ja muut antavat lisää esimerkkejä funktionaalisten kielten käytöstä järjestelmäohjelmoinnissa. [2]

2 Housen ominaisuuksia

House on IA32-prosessoriarkkitehtuurin päällä toimiva, Haskell-kielellä kirjoitettu pieni käyttöjärjestelmä, joka tarjoaa käyttäjälle mm. laiteajureita, ikkunointijärjestelmän, verkkoprotokollapinon ja komentokuoren. [2]

Houselle on kirjoitettu laiteajureita keskeytyskäsittelijälle, näppäimistölle, hiirelle sekä grafiikka- ja verkkokortille. [2] VGA-tekstitilan lisäksi käytössä on graafinen tila. [6]

Graafinen käyttöliittymä perustuu Noblen Gadgets-ikkunointijärjestelmään [9], joka käännettiin Haskell-kielelle Housea varten. [2] Noble mainitsee tavoitteekseen lisätä rinnakkaisajo laiskaan funktionaaliseen kieleen, jotta graafisen käyttöliittymän kuvaaminen olisi luonnollisempaa. [9]

Ethernet, IPV4, ARP, DHCP, ICMP (ping), UDP, TFTP ja TCP ovat kaikki tuettuja Housen verkkoprotokollapinossa. [6] [2] Hallgren ja muut kertovat, että verkkoprotokollapino on suurin yksittäinen Housea varten kehitetty osa. Toteutettu laiteajuri on tehty QEMU-emulaattorin[10] NE2000-verkkokorttia varten. He kutsuvat ratkaisuaan perinteiseksi ja laajennettavaksi. [2] Emulaattoriverkkokortin lisäksi myös Intel PRO/100-verkkokortille on ajuri. [6]

Komentokuoreen voidaan ladata ohjelmia ajettavaksi suojattuina käyttäjän prosesseina. Tämä tapahtuu joko siirtämällä a.out-binääritiedostoja TFTP:llä tai GRUB-käynnistyslataimen [11] avulla. [2]

House siis tarjoaa käyttäjälle keskeisimmät käyttöjärjestelmän toiminnot. Siihen on mahdollista kirjoittaa uusia laiteajureita ja näin laajentaa sen toimintaa entisestään. [6]

3 Käyttöjärjestelmän kehittämisen haasteet

3.1 Laitteiston kanssa toimiminen

Haskellin kaltaisen korkean tason kielen kanssa ongelmaksi nousee alimman tason laitteiston kanssa toimiminen. Hallgren ja muut ovat Housen tapauksessa kiertäneet ongelman tekemällä suoran laitteiston kanssa toiminnan siihen paremmin sopivalla kielillä ja abstrahoimalla laitteiston varsinaisen kernelin näkökulmasta. [2]

Diatchki ja Jones toteavat, että järjestelmäohjelmoinnissa joutuu tärkeitä osia kirjoittamaan C-kielellä tai assemblerilla Haskellin puutteiden vuoksi. Heidän mielestään ulkoiset funktiorajapinnat vaarantavat vahvan tyypityksen ja tyyppien turvallisuuden. [12]

Derrin ja muut väittävät, että yksinkertaisen ja tehokkaan ajonaikaisen järjestelmän päälle rakennettu kerneli abstrahoi laitteiston siinä määrin, että matalan tason ohjelmakoodista on helpompi tehdä päätelmiä. [4]

3.2 Rinnakkaistaminen

Käyttöjärjestelmät ja graafiset käyttöliittymät on helpompi toteuttaa rinnakkaisen koodin avulla. Rinnakkaisuutta on lisätty Haskelliin mm. Concurrent Haskell -laajennuksella [13].

Hallgrenin ja muiden mukaan rinnakkaisuus on järkevän kernelin toiminnalle välttämätöntä. Itse kernelin koodi, useat käyttäjän prosessit sekä keskeytykset joutuvat jakamaan laskenta-ajan. Housessa tämä on ratkaistu Concurrent Haskellilla. Jokainen rinnakkainen prosessi on oma säikeensä. Samoin keskeytys voidaan käsitellä omassa säikeessään. [2]

4 Housen tekninen perusta

House pohjautuu hOp-käyttöjärjestelmän matalan tason toteutukisista huolehtivaan ajonaikaiseen järjestelmään, joka on karsittu versio GHC:n omasta järjestelmästä. Lisäksi laitteiston käyttö on abstrahoitu H-laitteistomonadin taakse. [2]

Kaikki Haskell-koodi käännetään normaaliin tapaan GHC:llä ja tämän jälkeen luodaan GRUB-käynnistyslataimelle sopiva käynnistyskuva. Myös uusien ohjelmien kirjoittaminen ja varsinainen käyttöjärjestelmän laajentaminen tapahtuu Haskellilla. [2]

4.1 GHC:n ajonaikainen järjestelmä

House-ydin pohjautuu Carlierin ja Bobbion hOp-mikrokerneliin. Sen perustana on Glasgow Haskell Compiler -kääntäjän (GHC) karsitusta ajonaikainen järjestelmä (runtime system, RTS). Myös Housen toiminta perustuu tämän ajonaikaisen järjestelmän tarjoamalle roskienkeruulle ja rinnakkaistamiselle. [2]

Ajonaikainen järjestelmä on kirjoitettu C-kielellä. [6] Samoin suoraan laitteiston kanssa toimiva koodi on kirjoitettu C:llä ja assemblerilla. [2] GHC:n ajonaikaisen järjestelmän C-koodia on muokattu, jotta järjestelmä saadaan ajettua suoraan laitteistolla ja muistin sivutus alustettua. [2] Hohmuth ja Tews toteavat, että moderneja kieliä käytettäessä taataan, että testit tai toimenpiteet tehdään. He kuitenkin arvostelevat Housen matalan tason C-koodia, sillä sen toiminta ei ole samalla tavalla varmaa. [7]

Muistin kannalta ajonaikainelle järjestelmälle varataan erillinen C-pino sekä -keko. Haskellille on oma kekonsa erikseen myöhemmin muistissa. [2]

4.2 Laitteistomonadi H

Ajonaikainen järjestelmä ja matalan tason laitteiston kanssa viestiminen on piilotetaan rajapinnan taakse, jota kutsutaan nimellä H. Hallgren ja muut kuvaavat sen olevan eräänlainen erikoistunut IO-monadi. Se toimii lähempänä laitteistoa kuin IO-monadi ja tarjoaa sen lisäksi mahdollisuuden päästä käsiksi virtuaalimuistiin sekä ajaa käyttäjätilassa binääritiedostoja. [2]

H on toteutettu suoraan IA32-arkkitehtuurin päälle. Se on kirjoitettu pääasiassa Haskellilla. Sitä tukee C-koodi sekä pieni määrä assembleria, joita käytetään Foreign Function Interfacen (FFI) avulla. Tätä toteutusta voi käyttää turvallisesti Concurrent Haskellin kanssa. [2]

H on siis matalan tason monadinen laitteistorajapinta, jonka päälle voidaan toteuttaa käyttöjärjestelmä Haskell-kielellä. Rajapinta tarjoaa muistinhallinnan, käyttäjän prosessien ajamisen ja matalan tason laitesiirrännän (I/O). [2]

Hallgrenin ja muiden mielestä H on pieni ja yksinkertainen, ja tämän takia sen toiminta on helppo formalisoida. Tämä johtaa heidän tavoitteeseensa tehdä kerneli, joka on turvallinen, eikä voi pilata Haskellin kekomuistia. [2]

Hallgren ja muut esittelevät tarkemmin, kuinka H toteuttaa fyysiset sivut, virtuaalimuistin, käyttäjän ohjelmien ajamisen, I/O:n ja keskeytykset sekä rinnakkaisuuden. [2]

5 Yhteenveto

Haskell tarjoaa mahdollisuuden kirjoittaa käyttöjärjestelmä hyödyntäen funktio-ohjelmoinnin keinoja. Käyttöjärjestelmän toiminta on mahdollista kuvata Haskellin kaltaisella funktionaalisella ohjelmointikielellä abtraktion ja monadien avulla.

House perustuu laitteiston abstrahointiin H-laitteistomonadin avulla. Tämän perustan päälle on voitu rakentaa kaikki käyttöjärjestelmän tarjoamat palvelut.

Korkeamman tason kieliä käyttämällä ja erityisesti funktionaalisia kieliä pyritään käyttämään käyttöjärjestelmän tekemiseen, jotta sen toiminta voitaisiin varmistaa formaalisti. [2]

Derrinin ja muiden mukaan Haskellin kaltaiset korkean tason kielet ovat rajoittuneita käyttöjärjestelmän toiminnallisuuden toteuttamisessa. Toisaalta heidän mielestään Haskellilla kirjoitetut käyttöjärjestelmät ovat osoittaneet kykynsä mallintaa käyttöjärjestelmässä tarvittavaa toiminnallisuutta. [4]

Housen tapauksessa funktio-ohjelmoinnin taakse on onnistuttu piilottamaan varsinainen laitteiston kanssa toimiminen rajapinnan avulla. Tämä helpottaa käyttöjärjestelmän laajentamista sekä sovellusten kehittämistä.

Lähteet

[1] Jonathan Shapiro, *Programming language challenges in systems codes: why systems programmers still use C, and what to do about it*, PLOS '06: Proceedings of the 3rd workshop on Programming languages and operating systems, p. 9, 2006, San Jose, California

[2] T. Hallgren, M. P. Jones, R. Leslie, ja A. Tolmach, *A principled approach to operating system construction in Haskell*, ICFP '05: Proceedings of the tenth ACM SIGPLAN international conference on Functional programming, p. 116-128, 2005, New York, NY, USA

[3] K. Elphinstone, G.n Klein, P. Derrin, T. Roscoe ja G. Heiser, *Towards a practical, verified kernel*, 11th HotOS, 2007, San Diego, CA, USA

[4] Derrin P., Elphinstone K., Klein G., Cock D. ja Chakravarty M. M., *Running the manual: an approach to high-assurance microkernel development*, Proceedings of the 2006 ACM SIGPLAN Workshop on Haskell, 2006, Portland, Oregon, USA

[5] hOp-mikrokernelin dokumentaatio, saatavilla verkosta <http://etudiants.insia.org/ jbobbio/hOp/>, haettu 6.4.2008

[6] Housen lähdekoodi ja dokumentaatio, saatavilla verkosta <http://programatica.cs.pdx.edu/House/download/House-0.8.tar.bz2>, haettu 6.4.2008

[7] M. Hohmuth ja H. Tews, *The VFiasco approach for a verified operating system*, Proc. 2nd ECOOP Workshop on Programming Languages and Operating Systems, 2005, Glasgow, UK

[8] Tim Carstens, Kinetic-käyttöjärjestelmän dokumentaatio, saatavilla verkosta <http://www.ninj4.net/kinetic/>, haettu 8.4.2008

[9] R. Noble, *Lazy Functional Components for Graphical User Interface*, PhD thesis, 1995, University of York

[10] F. Bellard, *QEMU*, saatavilla verkosta <http://fabrice.bellard.free.fr/qemu/>

[11] GRUB-käynnistyslatain, saatavilla verkosta <http://www.gnu.org/software/grub/>, haettu 8.4.2008

[12] Diatchki I. S. ja Jones, *Strongly typed memory areas programming systems-level data structures in a functional language*, Proceedings of the 2006 ACM SIGPLAN Workshop on Haskell, p. 72-83, 2006, Portland, Oregon, USA

[13] Peyton Jones S., Gordon A. ja Finne S. *Concurrent Haskell* Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, p. 295-308, 1996, St. Petersburg Beach, Florida, United States