HAMK
Hämeen ammatti-
korkeakoulu

Ohjelmointikäytäntöjen merkitys energiansäästössä muistin käytön osalta

Sähkö- ja automaatiotekniikan opinnäytetyö

Sähkö- ja automaatiotekniikka

Syksy 2025

Joonas Rope

HAMK
Hämeen ammatti-
korkeakoulu

Sähkö- ja automaatiotekniikka Tiivistelmä
Tekijä Rope Joonas Vuosi 2025
Työn nimi Ohjelmointikäytäntöjen merkitys energiansäästössä muistin käytön osalta
Ohjaaja Henttonen Juhani

Työn tavoitteena on luoda ohjelma C++ ohjelmistokielellä kokeilemaan eri ohjelmistokäytäntöjen vaikutuksia keskusmuistin käyttöön energiansäästön näkökulmasta. Ohjelma tulee pyörimään Linux ympäristössä sen antamien tarkkojen mittaustapojen vuoksi.

Ongelma on maailmanlaajuinen koska ohjelmien ja automaation määrä kasvaa jatkuvasti ja esimerkiksi taustaohjelmat ja automaattisesti ajettavat ohjelmat ovat lisääntyneet huomattavasti. Uudempi fyysinen rauta kuluttaa yleensä enemmän energiaa ei vähemmän kuin vanha ja ohjelmistojen monimutkaisuus lisää tarvetta entistä tehokkaampiin koneisiin. Jos saadaan energian käyttöä vähennettyä pelkällä ohjelmistopäivityksellä niin se automaattisesti parantaa energiatehokkuutta tekemättä kalliita rautapuolen päivityksiä.

Sisältönä on Linux puolen teoria keskusmuistin kartoittamiseen proc/PID tiedostosta ja sen taulukointi ohjelmasta saaduilla tuloksilla. Ohjelma käsittää yksinkertaisia laskutoimituksia tehtynä eri ohjelmointitavoilla. Lukujen järjestämistä algoritmilla ja ilman sekä alkioiden järjestämistä eri ohjelmointitavoilla.

Johtopätöksinä eri ohjelmointitavoilla oli vähemmän merkitystä kuin alunperin odotettiin vain algoritmin ja perinteisen lajittelu metodin välillä oli huomattava ero. Syynä saattoi ola kääntäjän ja käyttöjärjestelmän erittäin hyvä etukäteisoptimointi.

Työssä olisi voinut käyttää yhden ohjelman sijasta useita pienempiä jolloin kokonaisvaikutus keskusmuistiin olisi ollut pienempi. Toisaalta se olisi vienyt tarkoitusta työltä jossa yritettiin päästä rasittamaan keskusmuistia ja pääsemään vähemmän energiatehokkaaseen swap osioon.

Avainsanat Virtuaalinen muisti, keskusmuisti, C++, ohjelmointikäytännöt.
Sivut 26 sivua ja liitteitä 11 sivua

HAMK
Hämeen ammatti-
korkeakoulu

Electrical and automation engineering
Author Rope Joonas
Subject Importance of programming practices to memory usage from energy saving perspective
Supervisors Henttonen Juhani

Abstract
Year 2025

Aim of the thesis was to create c++ program for testing purposes. Main aim was to test Random Access Memory from the point of view of energy saving. Program will be made and run on Linux for specific files and accuracy needed. Especially proc/PID was extremely helpful.

Background for the thesis comes from my own interest in programming and energy enthusiam. Energy saving or green coding in computers is necessary for the increase of automation and background programs. Often new hardware use more energy than older and new programs are much more complex and numerous than previous versions. If one were able to reduce used energy just by software update. Then you could delay buying new hardware or immidiately drop energy costs without user even noticing.

Content is one C++ program that takes snapshots 20 times during each part of the program. From these shots can be created file telling how much Random Access Memory is being used at any part of the program. Sanpshots are taken from /proc/PID file. After snapshots have been taken charts can be made and analyzed. Different parts of program are sorting with and without algrithm. Simple calculations in different programming methods and finding numbers in different ways. All numbers are first generated in random order.

Outcome was less than desirable only std::sort and traditional sorting showed large difference. Almost all other studied methods had virtually no difference from each other. Main reason could have been very optimised GNU compiler and very optimizing operating system Linux.

As conclusion program could have been more efficient if it were separated for many small programs instead of one large one. At that point overall impact for Random Access Memory could have been smaller. Then again that would have been counter productive since the whole point was to stress Random Access Memory until swap partion would have been used.

Keywords Virtual memory, RAM, C++, green coding
Pages 26 pages and appendices 11 pages

Sisällys

Kuvat, taulukot ja kaavat

Kuvia on 8 kpl ja taulukoita 3 kpl

Liitteet

1 Johdanto

Työn tarkoituksena on kehittää ohjelma rasittamaan tietokoneen keskusmuistia. Pyrkimyksenä on päästä lähelle keskusmuistin maksimi kokoa, jotta tietokone joutuu kirjoittamaan tietoa kovalevylle keskusmuistista. Tämä toiminto on energiaa kuluttavaa sekä kovalevyn että keskusprosessorin toimesta. Koska keskusmuistia ei itsessään voi optimoida energiansäästön osalta kuin vähäisesti, jää ainoaksi vaihtoehdoksi tutkia kuinka se vaikuttaa kun mennään lähelle maksimirajoja ohjelmalla.

Syy tutkimiselle lähti avoimen lähdekoodinkeskusteluista, joissa pyrittiin optimoimaan muistia nimenomaan keskusmuistin kautta. Kuinka olisi ollut mahdollista vähentää maailmanlaajuisesti energiankulutusta vain ohjelmoimalla muistinkäyttöä tehokkaammaksi. Esimerkkinä oli youtuben konesalit, joiden energiankulutus on mittava. Entä jos saadaan pelkästään ohjelmoimalla laskettua servereiden energiakulutusta edes promillen ilman että uusitaan rautaa.

Käyttöjärjestelmäksi valikoitui Linux ympäristö ja C++ ohjelmointikieli, josta löytyy omaa kokemusta ja tarvittavat työkalut. Osaa työssä käytettävistä tiedoista ei kerätä Windows käyttöjärjestelmässä. Varsinkin Linuxin antama vapaa mahdollisuus sorkkia kaikkia tiedostoja ja kerätä niistä tietoa vapaasti oli tärkeä ominaisuus. Ohjelmointikielen tuli olla käännettävä ettei sen ajaminen häiritsisi tehtävän ohjelman tuloksia. Sen tuli olla myös tarpeeksi stabiili ettei siinä olisi ohjelmointikielen kehittäjien tekemiä virheitä. Tämän lisäksi Linux on kirjoitettu isolta osalta C kielellä, joten niiden yhteensovittaminen oli helppoa ja antoi mahdollisuuden ja pakon käyttää välillä molemmpia kieliä yhtäaikaa.

Aluksi teoriapuolella keskitytään Linux:ssa tarvittaviin ohjelmiin, komentoihin ja miten energiakulutusta edes voi mitata. Ensin selvitetään kuinka tietokone, Linux ja C++ käsittelevät muistia. Eli mitä eri muistilajit ovat ja niiden mahdolliset vaikutukset energiakulutukseen. Teoriassa käsitellään myös miksi valittiin juuri GNU kääntäjä ja sen tuottamat mahdollisuudet optimoinnissa.

Fyysiseltä puolelta tuodaan esiin kuinka eri muistit käyttäytyvät ja toimivat keskenään. Lähtökohtaisesti mikään muisti ei toimi yksin vaan on tärkeää havainnoida milloin se keskustelee muiden muistien kanssa. Mikä on se hetki kun keskusmuistista kirjoitetaan tietoa kovalevylle järjestelmän pelastamiseksi ja uhrataan energiankäytössä ja nopeudessa.

Ohjelmapuolella keskitytään haasteisiin kuinka ohjelmia edes voidaan mitata Linux:ssa ja miten se voidaan tehdään yksittäisen ohjelman osalta. Ongelmakohtina on ohjelman ajon aikainen seuranta ja niiden tarkuudet. Työssä päädytään käyttämään itsetehtyä ohjelmaa joka mittaa miilisekuntitasolla tavoitelluissa kohdissa. Itsetehtdyistä ohjelmista käydään läpi miten ja miksi ne toimivat sekä niiden tarkoitus.

Varsinainen pääohjelma eli muistinkäyttöä kokeileva ohjelma käydän osin ohjelmointitasolla läpi ja kerrotaan kuinka se toimii. Samalla hahmotetaan ongelmaa, joka syntyy kun ohjelman pitäisi seurata itseään ennenkuin se käynnistetään. Suoritettavat laskutoimitukset ovat yksinkertaisia tarkoituksella. Mitä monimutkaisempi ohjelma sitä enemmän tuntemattomia muuttujia ja seurattavia muutoksia.

Tuloksissa käydään läpi millaisia ongelmia ajon aikana tuli ja etsittiin niihin mahdollisia syitä. Mahdollisia syitä löytyi aina ohjelmoijasta, käyttöjärjestelmään ja kääntäjään asti. Luettavuuden helpottamiseksi ne on esitetty taulukko muodossa jossa verrataan järjestelmästä saatuja tietoja ja pohditaan niiden järkevyyttä. Ohjelman tulokset on jaettu kolmeen eri vaiheeseen eli etsimiseen, järjestelyyn ja laskemiseen. Siinä tulee ilmi kuinka paljon kukin vaihe käytti eri muistilajeja.

Lopuksi tarkastellaan ovatko tulokset järkeviä ja niihin vaikuttaneita seikkoja sekä kuinka hyvin tulokset vastasivat odotuksia ja mitä poikkeamia esiintyi.

2 Energiakulutuksen ja muistinkäytön välinen yhteys kirjallisuudessa.

Kirjallisuudessa on käsitelty kattavasti muistinhallintaa ohjelmoinnin osalta ja varsinkin C++ osalta. Energiansäästöjen kannalta tarkkoja tutkimuksia missä katsotaan virrankulutuksen yhteyttä muistin käyttöön ei löydy yhtä paljon. Kuitenkin yleisellä tasolla energiankulutuksen ja ohjelmoinnin yhteyteen on tehty tutkimuksia.

Yhtenä esimerkkinä oli Javan ja C++ välinen tutkimus vuodelta 2006 Aspects of Memory Management in Java and C++. Tutkimuksessa mainittiin C++ vaikeasta muistinhallinnasta ja sen virhealttiudesta etenkin pointtereiden osalta. (Gayathri, 2015)

Toisena muistin käytön vertailu C++ ja Pytonin välillä vuonna 2015 (Comparative Analysis of C++ and Python in Terms of Memory and Time). Tutkimuksessa vertailtiin ohjelmointikielien muistinkäyttöä ja toteutusnopeutta yksinkertaisien ohjelmien avulla.

Energiansäästön osalta löytyi lopputyö vuodelta 2013 Energy Efficient Programming, jonka sisältö keskittyi pääosin yleisellä tasolla ohjelmoinnin avulla saataviin energiansäästöihin ja CO2 päästöjen vähentämiseen.

Vuodelta 2016 löytyi kattava tutkimus (A study of the influence of software and hardware features on program energy) ohjelmointikäytäntöjen vaikutuksesta energiankulutukseen. Tutkimuksessa kerättiin tietoa 58 eri ohjelmasta, kuinka energiaa säästävät toimintatavat eroavat muista.

Vuodelta 2012 löytyi tutkimus Modeling Power Consumption of High Performance Computer Clusters Mario Cairelta. Tutkimuksessa tutkittiin /proc tiedostosta kuinka paljon keskusprosessorit kuluttivat energiaa.

3 Muistinkäytön mittaaminen Linux käyttöjärjestelmässä

3.1 Muistinkäytön perusteet Linuxissa

Käyttöjärjestelmä jakaa muistia keskusprosessorin muistin, keskusmuistin ja swap osion välillä. Linux jakaa käyttämätöntä muistia ohjelmien välillä automaattisesti parantaakseen ohjelmien toimimista ja täten vapaan muistin määrä ei kerro oikeasti vapaan muistin määrää. Muistin tarpeen noustessa Linux vapauttaa toissijaisilta ohjelmilta muistia tarpeen mukaan. (Kernel, n.d.)

Riippuen sivun käytöstä Linux käsittelee sitä eri tavoin. Välimuistin voi vapauttaa milloin vain koska sen tiedot ovat tallella muualla. Kernelin ja DMA:n tietoja ei voi vapauttaa ennen kuin niiden käyttäjä vapauttaa ne. (Kernel, n.d.)

”Linux vapauttaa ja uudelleen käyttää sivuja asynkronisoidusti tai synkronisoidusti riippuen järjestelmän tilasta. Kun järjestelmää ei ole ladattu suurin osa muistista on vapaana ja muistitarve täytetään tarpeen mukaan. Kuormituksen lisääntyessä vapaiden sivujen määrä laskee ja kun se menee tarpeeksi alas Linuxin aliohjelma alkaa skannaamaan sivuja ja vapauttaa muistia mistä pystyy. (Kernel, n.d.)

3.1.1 Page (muistisivu)

Linux Kernel dokumentaation mukaan: ”fyysisen järjestelmän muisti jaetaan useisiin muistisivuihin, joiden koko riippuu arkkitehtuurista ja voidaan valita Kernelin kääntämishetkellä. Fyysinen muisti kartoitetaan virtuaaliseksi muistiksi, joka luo page tablehin, jota ohjelmat käyttävät fyysisen muistin käsittelyyn. Alimman tason tablet käsittävät fyysiset muistiosoitteet, joita ohjelmat käyttävät. Ylimmän tason tabletit sisältävät alempien tasojen fyysiset muistiosoitteet. Keskusprosessorin käyttää rekisteriä kääntäessään muistiosoitteita.” (Kernel, n.d.)

3.1.2 Isot muistisivut (Huge Pages)

Keskusprosessorin osalta hitautta tuo muistiosoitteiden kääntäminen, koska muistiosoitteiden kääntäminen on hitaampaa kuin keskusprosessin nopeus. Prosessorin syklien säästämiseksi keskusprosessori pitää oma muistiosiota (TLB) Ohjelmat, jotka varaavat paljon muistia kärsivät tehokkuudessa, kun TLB toimii liian hitaasti. (Kernel, n.d.)

Moderneissa keskusmuisti arkkitehtuureissa voidaan kartoittaa isompia sivumuistin kokoja. X86 voidaan kartoittaa 2 Megan ja 1 Gigan kokoisia muistisivuja. Linuxissa isompiin muistisivuihin viitataan sanalla “iso”. Ne vähentävät TLB painetta ja parantavat keskusprosessorin suorituskykyä. (Kernel, n.d.)

3.1.3 Alueet

“Usein laitteisto antaa rajoitteita kuinka fyysiseen muistiin päästään käsiksi. Esimerkiksi DMA:n vuoksi laitteisto ei pääse käsiksi kaikkeen käytettävissä olevaan muistiin. Toiseksi laitteiston muisti saattaa ylittää käytettävissä olevan virtuaalisen muistin ja ylimääräisiä toimia tarvitaan, että muistiin päästään käsiksi.” (Kernel, n.d.)

3.1.4 Nodet

Moni ytimisissä koneissa muisti on järjestelty riveiksi, joilla on eri saatavuus latenssi riippuen etäisyydestä prosessoriin. Jokainen rivi on node ja sillä on oma itsenäinen muistinhallinta alajärjestelmä.” (Kernel, n.d.)

3.1.5 Sivun välimuisti

“Fyysinen muisti heittelee paljon ja yleisesti mustiin lisätään tietoa lukemalla se tiedostoista. Kun tieto on luettu se jää sivun välimuistiin, jolloin tiedostoa ei tarvitse koko ajan lukea. Samoin tapahtuu, kun tiedostoon kirjoitetaan tekstiä.” (Kernel, n.d.)

3.1.6 VSZ virtuaalinen muisti

“Linux tukee virtuaalista muistia eli kovalevyn käyttämistä keskusmuistin jatkeena, jotta käytettävän muistin tehokkuus kasvaa vastaavasti. Kernel kirjoittaa tällä hetkellä käyttämättömiä muistiblokkeja kovalevylle, jotta muistia voidaan käyttää toiseen tarkoituksessa, kun prosessi tarvitsee alkuperäistä sisältöä uudestaan. Se luetaan takaisin muistiin.” (Kernel, n.d.)

”Virtuaalisen muistin koko Linuxissa eli VSZ on muistin koko jonka Linux on varannut ohjelmalle mutta ohjelma ei välttämättä käytä kaikkea siitä muistista. Muistia varataan Linuxissa tarpeen mukaan.” (Reynolds, 2021)

3.1.7 RSS (Resident set size)

RSS kertoo kuinka monta ”sivua” ohjelma varaa muistista mutta ottaa huomioon myös jaetut kirjastot sekä dynaamisesti linkitetyt kirjastot. Wikipedian mukaan: “RSS on se osa muistista, jonka prosessi käyttää keskusmuistista.” (Wikipedia, 2024 -a)

”Linux configin mukaan RSS on se osa muistista, jota käytetään lataamaan kaikki muistisivut mukaan luettuna jaetut kirjastot.” (Reynolds, 2021)

3.1.8 PSS Suhteellinen koon määritys

”PSS kertoo hieman tarkemmin minkä verran ohjelma vie muistia mutta ei edelleenkään allokoi jaettuja tiedostoja tai linkitettyjä kirjastoja suoraan käytettyihin resursseihin. Eli jos samaa kirjastoa käyttää kaksi ohjelmaa niin sen keskusmuistin osuus jaetaan kahdella. Eli vähän käytetyt kirjastot keskusmuistissa kokeiltavalla ohjelmalla saavat laskennallisesti puolet, vaikka niitä kutsutaan kerran tai ei kertaakaan. Pss on Linuxille ominainen arvo.” (Wikipedia, 2024-b)

Wikipedian mukaan PSS näyttää kuinka paljon ohjelma käyttää muistista mukaan luettuna jaettujen kirjastojen suhteellinen osuus. (Wikipedia, 2024-b)

3.1.9 Mittauksen haasteet nykyisissä Linux käyttöjärjestelmissä

Kiitos PSS ja RSS:n tarkkaa muistin käyttöä on lähes mahdoton löytää jaettujen kirjastojen vuoksi. Jaetut kirjastot jaetaan kaikille niitä käyttäville ohjelmille tasaisesti koska ne ovat vain luku muodossa keskusmuistissa.

Toiseksi /proc tiedosto ei anna tarkkoja lukuja muistinkäytöstä se ei ole luotu sitä varten. /proc/pid tiedosto taas luodaan vasta kun prosessi on saanut oman ID numeronsa. Täten sivua ei myöskään päivitetä ennen kuin sitä kutsuu, jokin toinen ohjelma.

Linux järjestelmässä filesystemin koko on 4096 kB, joka on muistisivun koko.

3.2 Muistinhallinta C++ ohjelmoinnissa

C++ muistia hallinnoidaan kolmella eri tavalla automaattisesti (stack allocation), staattisesti (static allocation) ja dynaamisesti (static allocation).

Automaattisesta muistinhallinnasta todetaan Vassevin ja Paquetin mukaan seuraavasti. Se rajoitetaan käsittelemään vain paikallisia muuttujia. (Vassev & Paquet, 2006)

”Staattista allokaatiota käytetään globaaleihin ja staattisiin muuttujiin. Näiden muuttujat kestävät koko ohjelman suorituksen ajan. Ja tätä muistia ei voi käyttää muualla ohjelman ollessa päällä.” (Vassev & Paquet, 2006)

3.3 Pino (stack)

Pino on se osa muistia, jota käytetään lyhytaikaiseen säilytykseen Lifo periaatteella. Koska jokainen funktiokutsu luo uuden pinon on se hyödyllinen muistinhallinnassa. Pino vapauttaa täten muistia keskusmuistista, kun funktiokutsu on ohi. Pinon koko on rajallinen ja se voi helposti vuotaa. (Computer science wiki, 2023-b)

3.4 Heap

Heap toimii dynaamisesti allokoitavana muistina. Toisin kun pinossa muisti on käytettävissä pidemmän aikaa eli siihen asti, kunnes ohjelma lopetetaan tai muistia tarkoituksella vapautetaan. Heapin muistia ei hoideta automaattisesti, joten ohjelmoijan on vapautettava se itse. Se on myös pinoa hitaampi. (Computer science wiki, 2023-a)

3.5 C++ ohjelmointikielen vaikutus muistinhallintaan

Tämä työ tehdään käyttäen C++20 standardia. Vaikka C++23 on tullut ulos ei sitä ole vielä integroitu hyvin GNU kirjastossa. Turvallisin olisi C++17 joka on paremmin integroitu kääntäjään. Toisaalta monet muistinhallintaan liittyvät parannukset tulivat jo C++17 standardissa ja ne ovat jo integroitu, vaikka käyttää C++20 standardia. (GNU, 2024)

3.5.1 Dynaaminen array vs staattinen array

”Vektorin muisti hoitaa itsensä automaattisesti ja laajentuu tarpeen mukaan. Muistin käytön osalta ne varaavat enemmän muistia, kun staattiset array:t tulevaisuuden muistintarvetta varten. Täten vektorin ei tarvitse uudelleen varata muistia muulloin, kun sen oma muisti on lopussa. Koko varatun muistin saa komennolla capacity(). Varatun muistin saa vapautettua komennolla shrink_to_fit().” (CPPReference, 2023-a)

”Staattisen Arrayn muisti luodaan kääntämisen aikana ja sitä ei voi muuttaa ohjelman käyntiaikana” (Geeks for geeks, 2023-a).

”Staattisen ja dynammisen arrayn välinen ero on, että dynaaminen array sijaitsee pinossa (stack) ja dynaaminen array sijaitsee heapissa. Toisena erona on milloin array muisti varataan stattinen array varaa muistin kääntämisen aikana ja dynaaminen array varaa käynin aikana. Siinä missä dynaamisen arrayn kokoa voi muuttaa tarpeen mukaan staattisen ei voi vaan se pysyy samankokoisena koko olemassaolo aikansa” (Geeks for geeks, 2023-a)

4 GNU kääntäjä

GNU kääntäjä on omien sanojensa mukaan: ” Kokoelma käyttöliittymiä C/C++, Fortranille, jne. Myös näiden ohjelmointikielien kirjastoille” (GNU,2024-a). Kääntäjän tehtävänä on luoda lähdekoodista konekielistä. Kyseessä on suosittu ja hyvin yleisesti käytetty kääntäjä vapaan

lähdekoodin ohjelmoinnin puolella. Sen pitkä kehityshistoria ja vakaus antavat mahdollisimman paljon räätälöinti mahdollisuuksia työtä varten.

Flageilla tarkoitetaan kääntämisen yhteydessä lisäkomentoja, joilla voidaan muokata käännettävää ohjelmaa. GNU:n osalta tärkeät kohdat ovat -O3, -O2, -O1 ja -O0 flagit. Näillä kohdilla mahdollistetaan tarkempi optimointi yleensä suuremmalla kääntämiseen kuluneella ajalla ja/tai suuremmalla ohjelmalla. (Nwachukwu, 2023)

Muut tarvittavat flagit ovat -std=c++20, joka mahdollistaa C++ standardi 20 käytön. Standardin määrittäminen on tärkeää, koska ei standardeissa ohjelma tulee kirjoittaa erillailla. Pääsääntöisesti uudempi standardi antaa helpomman tai kattavamman tavan tehdä ohjelmointia mutta ei välttämättä toimi vanhempien ohjelmien tai koneiden kanssa.

5 Eri muistit ja niiden käyttäytyminen tietokoneessa.

Muisti vaikuttaa keskusprosessorin kautta energian kulutukseen, vaikka RAM:illa onkin sama jännite koko ajan. Keskusprosessori joutuu tekemään enemmän töitä, minne se saa mahtumaan käsiteltävän tiedon. Swap osio ja muisti nimenomaan säilövät ohjelman käyttämiä muuttujia ja listoja, jotka ottavat muistilta paljon tilaa riippuen listan tai muuttujan koosta.

5.1 Keskusprosessori

”Keskusprosessorin tehtävänä toteuttaa tietokoneohjelman käskyjä kuten aritmeettisia, loogisia, ohjailevia ja I/O toimintoja”(Computer Science Wiki, 2022).

Keskusprosessorin muistissa on useita tasoja kuten L1, L2 ja L3. Pääasiassa niiden koko, nopeus ja sijainti vaihtelevat. L1 muisti on nopein mutta pienin ja se on jaettu kahteen osaan eli ohje osioon ja tieto osioon. L2 muisti on hitaampi kuin L1 taso mutta siinä on enemmän muistia. L3 tasolla on eniten muistia mutta se on hitain kaikista tasoista. Keskusprosessorin muistin määrä vaikuttaa nopeuteen mutta ei ole ainoa vaikuttava tekijä. (How-to-geek, 2023)

5.2 Keskusmuisti

RAM eli kuskusmuisti on väliaikainen talletustila kovalevyn ja keskusprosessorin välillä. Ohjelmat lataavat tietoa keskusmuistiin, jota keskusprosessori käyttää laskutoimituksiin.

Keskusmuistin tehtävänä on olla nopeampi paikka hakea tietoa kuin kovalevy. (Crystal, 2023)

5.3 Swap osio

”Linux jakaa fyysisen keskusmuistin paloiksi, joita kutsutaan page nimellä. Swap osio on etukäteen varattu kovalevyn osio, jonne voidaan kopioida tietoa keskusmuistilta, jotta voidaan vapauttaa keskusmuistista tilaa. Swap osion ja keskusmuistin yhteenlaskettua muistia kutsutaan virtuaaliseksi muistiksi.” (Gary, 2007)

6 Energiankulutuksen mittaamisen haasteet yksittäisen ohjelman osalta

Energian kulutuksen osalta mittaamisessa tulee olla käytössä wattimittari tai jokin muu tapa suoraa mitata kuinka paljon virta tai jännite muuttuu. Niitä ei ole käytössä tätä työtä tehdessä, joten ainoa keino pohtia energian kulutusta on mitata kuinka kauan ohjelman eri vaiheiden suorittaminen vie ja kuinka paljon ne kuormittavat keskusprosessoria.

Vaikka kannettavan tietokoneen akun energian kulutuksen mittaaminen onnistuukin eri ohjelmilla kuten powertop, lm_sensors, tlp_stat ja muilla niin ne eivät ole tarkkoja. Tässä työssä osa vaiheista on niin nopeita, ettei ilman ulkopuolista wattimittaria voida laskea arvoja. Pöytätietokoneen osalta energia kulutusta ei pystytä seuraamaan lainkaan. (Perez, 2024)

6.1 Turbostat

”Turbostat ohjelma kerää prosessorin topologian, tajuuden, aktiivisuuden, lämpötilan ja energiankulutuksen X86 prosessoreissa.” (Debian, 2024)

Ohjelman keräämien tietojen tarkkuus vähenee alle sekunnin näytteenottovälissä. Koska tässä työssä katsotaan kuinka paljon optimointi ja algoritmit vaikuttavat yksinkertaisissa funktioissa on näytteenottoväli millisekunnissa. Täten mittaus tarkkuus kärsii mutta jonkin näköinen mittaustulos on saatava tutkimusta varten. (Debian, 2024)

6.2 Energiankulutuksen seuraaminen ja laskeminen Linux käyttöjärjestelmissä

Kannettavien tietokoneiden energian kulutuksen seuraaminen tarkasti on vaikeaa ja koska tulosta pitää verrata laskettuun arvoon. Tässä osiossa käydään muutama matemaattinen laskentatapa ja eri mittaus ohjelmat, joita voidaan käyttää. Konsensuksena eri tutkimuksissa näyttää olevan, että tietokoneen virrankulutus mitataan suoraan virransyötöstä ja lasketaan eri valmistajien datalehtien tietojen avulla. Varsinkin kun virrankulutukseen vaikuttaa laitteiston kokoonpano, käyttöjärjestelmä ja hetkellinen kuormitus. Hyvänä puolena Linuxissa on muistin tarkka seuraaminen ja täten suhteellisen tarkat arvot.

7 Koe ohjelmien rakenne ja vaikutus muistin käyttöön.

Koeohjelman suunnittelemisessa suuri osa ajasta meni etsiessä, kuinka lokitiedot saa synkronoitua /proc/pid tiedoston tiedon kanssa. Ongelmana oli, että PID:tä eli prosessi ID:tä ei ole ennekuin ohjelma käynnistetään ja Linuxissa ei myöskään voi itse etukäteen määrittää PID:iä etukäteen. Lopulta ainoa vaihtoehto oli aloittaa ohjelma ja heti alussa kysyä mikä sen PID oli ja parsia se kiinni tarvittaviin lauseisiin, että voitiin kysyä käyttöjärjestelmältä mikä on muistin käyttö.

Lokitietojen kirjoittamisen vaikutukset ohjelman tarkkuuteen ja nopeuteen. Siinä missä C++ on käännettävä kieli ja täten nopeampi kuin Bash, joka on tulkattava kieli niin ohjelman suoritusnopeutta hidasti komento system(). Tämän komennon avulla täytyi löytää kaikki /proc/pid/stat, proc/pid/meminfo tiedostot ja kutsua niitä. Tällöin C++ ohjelma ei voinut luoda kääntämisen aikana kutsuttavia lauseita vaan ne piti luoda käynninaikana. Onneksi nuo lauseet tarvitsi luoda vain kerran. Oikea hidastus oli luoda varsinainen lokitiedosto joka kutsui system() komentoa useita kertoja putkeen, jotta saatiin kaikki tarvittavat tiedot lokeihin. Tarkkuus ajallisesti kärsi koska piti kirjoittaa lokitiedot tulkattavalla kielellä kovalevylle, joka on ajallisesti hitain prosessi koko kokeesta.

7.1 Linux järjestelmän kartoittava ohjelma (Bash)

Bash ohjelmaksi muodostui liitteessä 2 kuvattu keräysohjelma. Se pääasiassa kerää muistien koot eli keskusprosessorin L1, L2,L3 koot ja keskusmuistin koot ja muuta myöhemmin oleelliset asiat. Ohjelma myös poistaa turhia tietoja koska lshw komento antaa

erittäin tarkkoja tietoja tietokoneesta. Liitteessä 6 pöytäkoneen ja kannettavan tiedot kartoitus ohjelmasta.

Tämä tehdään vai kokeen uusimista varten eri koneilla, jotta voidaan verrata helpommin mikä on tietokoneen kokoonpano ja mahdolliset erot.

7.2 C++ ohjelma, joka kokeilee muistinkäyttöä

Tulokseksi saatu ohjelma kokeilee muistinkäyttöä kolmella eri tavalla etsimällä, järjestämällä ja laskutoimituksilla. Tarkoitus on rasittaa mahdollisimman paljon keskusmuistia, jotta päästään mahdollisesti swap-osiolle ja energiankulutuksen arviointiin.

Aluksi ohjelma etsii oman PID numeronsa, koska ilman sitä ei voida kerätä lokitietoja. PID numero parsitaan kiinni eri lauseisiin, jotta niitä voidaan kutsua system() komennolla. Kuvassa 1 kuva ohjelmasta.

Seuraavaksi tehdään apuohjelma (kuva 2), jota kutsumalla saadaan /proc/pid/stat ja /proc/pid/meminfo tiedostojen sisällöt tallennettua heti status.log tiedostoon päivämäärän ja ajan kanssa. Tässä vaiheessa ei tehdä karsimista vaan tallennetaan kaikki tiedot. Tarkoituksena on minimoida kuluva aika, keskusprosessorin ja keskusmuistin kuormitus.

Kuva 1 Koodi prosessi ID:stä

```
//get program ID to tempfile
void ID(){
        system("pidof Memtest.exe > temp.log");
}

std::string process_ID(){
        // Make string for log file to be read later, (ifstream reads ofstream writes)
         std::string vlogfile;
         std::ifstream logfile;

         // Open previously made log file and stream it in temporary log
         logfile.open("temp.log");
         std::stringstream streamlog;

         // read file's buffer contents into streams
         streamlog << logfile.rdbuf();

         // close file handlers
         logfile.close();

         // convert stream into string
         vlogfile   = streamlog.str();

         //Make /proc/PID/status file readable now that we have process ID
         std::string procFile {"/proc/"};

         // Erase newline from the end of process ID (messes up further steps)
         vlogfile.erase(std::remove(vlogfile.begin(),vlogfile.end(),'\n'), vlogfile.end());



         return vlogfile;
}
```

Kuva 2 Koodi lokitietojen keräämistä varten testiohjelmasta

```
void logPrinter(int cycle, std::string process, std::string procFile, std::string procStatm, std::string procStat){
       std::ostringstream temp_stringstream;
       temp_stringstream << cycle;
       std::string s_cycle = temp_stringstream.str();


       //Tell what process is going on and what cycle it is for csv file

       system("echo -en Process: >> status.log");
       system(("echo " + process +" >> status.log").c_str());
       system("echo -en Cycle: >> status.log");
       system(("echo " + s_cycle + " >> status.log").c_str());

       //print out info from system status and add date.
       //Date gives Month,Day,Year,Hour,Minute,Seconds and milliseconds

       system("echo -en Date:          >> status.log");
        system ("date +%s.%3N  >> status.log");

       // converts bash command outputs to bash file status.log at same folder as this program reside. c_str() needed for reading string with bash
        system (("cat " + procFile + "  >> status.log").c_str());
        system (("cat " + procStatm + " >> status.log").c_str());
        system ("echo -en procStat:     >> status.log");
       system (("cat " + procStat + "  >> status.log").c_str());
}
```

Kolmantena vaiheena luodaan satunnaisluvuilla numeroita 1 – 200000 välillä vektoriin. Numeroiden kokonaismäärä on myös 200000 ja joka 20000 välein otetaan väliarvo

lokitiedostoon. Näistä numeroista etsitään 10000 numeron välein 20 kertaa löytyykö tietty numero järjestelemättömästä vektorista. Jokaisesta vaiheesta otetaan lokitiedot 20 kertaa jokaiselta kohdalta. Näistä luodaan toinen vektori johon kopioidaan satunnaislukuvektori ja järjestetään se pienimmästä suurimpaan std::sort komennolla. Lokitiedot otetaan ennen ja jälkeen järjestelyn. Koska std::sort algoritmia ei voi pilkkoa.

Seuraavassa vaiheessa etsitään järjestellystä vektorista 20 kertaa numeroita 10000 välein löytyykö numero.

Tämän jälkeen siirrytään array malliin. Ensin kopioidaan järjestelemätön vektori arrayihin. Tämän jälkeen etsitään arraysta 10000 välein löytyykö tiettyä lukua. Ja myös 10000 välein otetaan lokitietoja status.log tiedostoon.

Seuraava vaihe on järjestellä array perinteisellä tavalla, eli verrata kahta alkiota ja siirtämällä pienemmän alkuun. Siinä käytetään kahta sisäkkäistä for looppia ja 10000 luvun välein otetaan lokitietoja ulos.

Sama järjestely toistetaan mutta nyt käyttämällä std::sort komentoa array:lle. Lokitiedot otetaan ennen ja jälkeen toiminnon.

Seuraavassa vaiheessa etsitään sekä for loopin avulla että ilman sitä kuinka kauan aikaa menee hakea tiettyyn alkioon asti. Hypoteesina on että for loopin pitäisi viedä kauemmin.

Ohjelman viimeisessä osassa tehdään kuusi for looppia, joissa lasketaan yksinkertaisten kuvioiden pinta- aloja satunnaisluvuilla kuten kuvasta 3 näkyy. Ensin neliö, kolmio ja lopuksi ympyrä. Ensimmäinen versio tehdään kirjoittamalla kullekin pinta- alalle oma funktio, jota toistetaan 20 kertaa eri luvuilla. Seuraavaksi toistetaan samat mutta tehdään yksi funktio lisää, jonka tehtävä on tehdä pelkästään kertolasku. Jälkimmäisessä ohjelmassa kokeillaan onko (DRY) eli ei kannata toistaa koodia merkityksellinen.

Kuva 3 Yksinkertaisia laskuja suorittava ohjelma

```
#include "calculate_simple_shapes.h"

void calculate_pyramid_simple(int height, int width){
        auto pyramid_size = ((height*width)/2);
}

void calculate_square_simple(int height, int width){
        auto squareSize=height*width;
}

void calculate_circle_simple(int diameter){
        auto ((diameter*PI)/2);
}

auto multiplyTwoNumbers(auto height, auto width){
        return height*width;
}

void calculate_pyramid_with_helpfunction(int height, int width){
        auto pyramid_size = (multiplyTwoNumbers(height,width)/2);
}

void calculate_square_with_helpfunction(int height, int width){
        auto squareSize=multiplyTwoNumbers(height,width);
}

void calculate_circle_with_helpfunction(int diameter){
        auto ((multiplyTwoNumbers(diameter,PI)*diameter)/2);
}
```

Ohjelmasta tulee siis lopuksi noin 150000 rivinen status.log tiedosto.

7.3 Keskusprosessorin kuormitusta laskeva ohjelma

Ennen muistin käyttöä kokeilevaa ohjelmaa laitetaan powertop ohjelmisto päälle taustaprosessiksi (kuva 4). Powertopin komennolla: sudo turbostat –Summary –quiet –show Busy%,AVG_Mhz,PkgWatt –interval 0.001 –enable Time_of_Day_Seconds -out turbo_CPU.log &. Saadaan kaikkien prosessorien yhteenlaskettu aktiivisuus, keskimääräinen taajuus megahertsinä ja paketin kuluttama energia wateissa. Time_Of_Day_seconds antaa

päivämäärän nanosekunteina vuodesta 01/01/1970 kello 00:00:00 alkaen. Tämä on pakollinen vaihe. Koska muuten ohjelmasta saatavia tuloksia ei saa sykronoitua muistin käyttöä kokeilevan ohjelman kanssa. Niiden aloitusvaiheilla on eroa noin 200 ms. Ohjelmasta tulee noin 150000 rivinen lokitiedosto (kuva 5). Vaikka ohjelma on laitettu maksimi näytteenottovälille eli millisekuntiin sen antamat arvot ovat välillä 1-5 ms riippuen koneen rasituksesta.

Kuva 4 Turbostat ohjelman koodi

```
1 #!/usr/bin/bash
2
3 sudo turbostat --Summary --quiet --show Busy%,Avg_MHz,PkgTemp,PkgWatt --interval 0.001 -out turbo_test.log &
```

Kuva 5 turbostat ohjelman tulokset

```
Time_Of_Day_Seconds,Avg_MHz,Busy%,PkgWatt
1723127894.688041,390,14.47,23.46
1723127894.690389,339,20.46,14.22
1723127894.693068,382,15.34,11.32
1723127894.695439,979,36.98,22.44
1723127894.697385,466,21.06,25.26
1723127894.698971,343,12.86,26.40
1723127894.700407,359,15.72,20.53
1723127894.702904,341,20.59,6.48
1723127894.704445,393,11.97,29.94
1723127894.706083,384,17.84,9.35
1723127894.707513,356,14.08,22.34
1723127894.709591,342,17.62,6.84
1723127894.712090,347,17.21,11.82
1723127894.714660,341,14.69,17.74
1723127894.716094,424,13.46,20.43
1723127894.718275,342,21.72,14.97
1723127894.719672,394,11.19,22.63
1723127894.721129,402,16.53,11.65
1723127894.723705,341,15.61,13.74
1723127894.726085,349,18.87,11.92
1723127894.727523,391,12.08,35.91
1723127894.728800,770,33.50,14.43
1723127894.730196,526,23.01,14.59
1723127894.731669,353,17.42,25.21
1723127894.734319,487,27.99,5.97
1723127894.735623,560,16.43,39.97
1723127894.737851,409,24.68,8.22
1723127894.739183,409,11.90,27.17
1723127894.740569,370,16.36,24.09
1723127894.742021,352,15.99,10.59
1723127894.744573,367,18.87,11.98
1723127894.746019,371,12.77,22.10
1723127894.748325,348,18.49,13.88
```

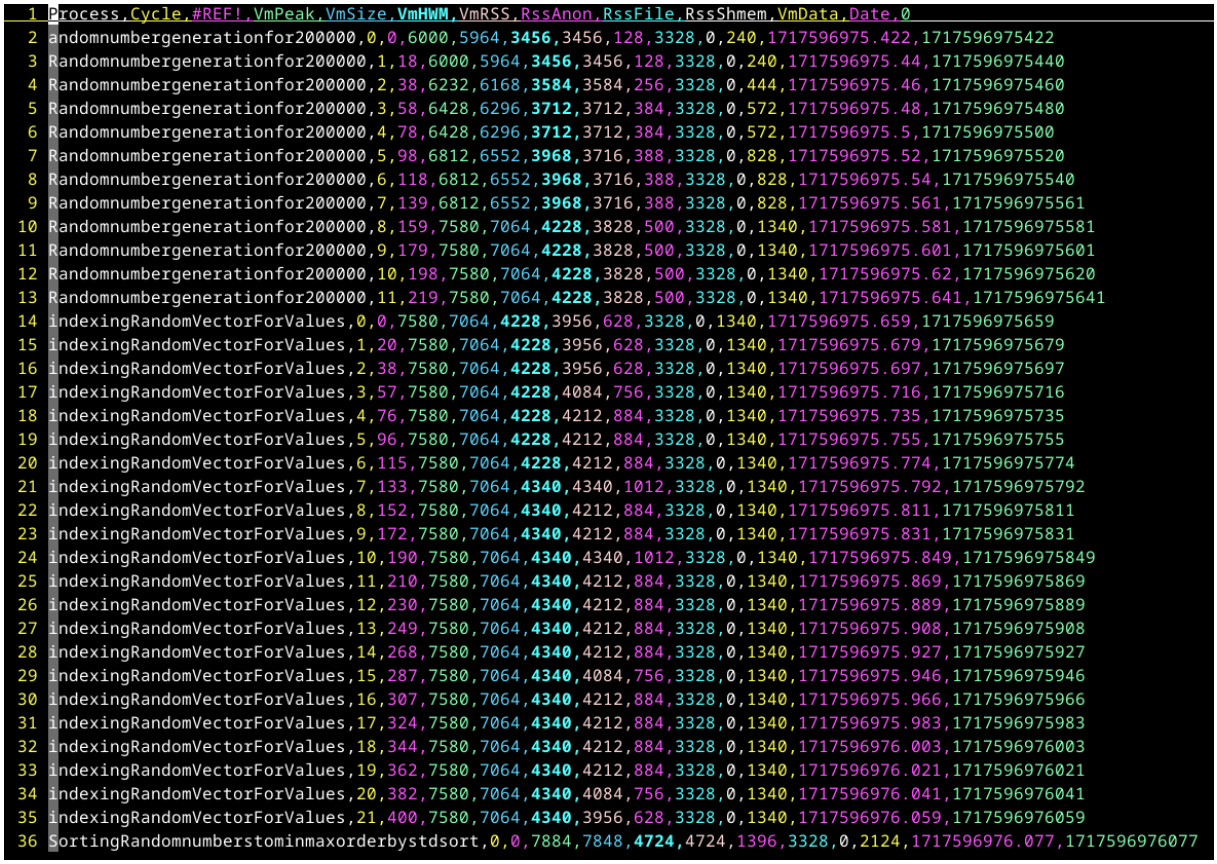
7.4 Lokin karsimisohjelma

Kun status.log tiedosto on valmis. Se käsitellään log_trimmer bash ohjelmalla (kuva 6). Tarkoitus on muuttaa se csv muotoon helpomman käsittelyn vuoksi. Aluksi lokista kerätään vain tarpeelliset sarakkeet kuten päivämäärä, muisti, VmSize, Rss ja muut hyödylliset sarakkeet. Loppuohjelma muuttaa kaikki tiedot omiin sarakkeisiin ja erottaa ne pilkulla. Tehden siitä CSV muotoiseksi kuten kuvasta 7 näkyy.

Kuva 6 log_trimmer ohjelma

```
# Only take wanted info from status.log
#grep 'Date\|VmPeak\|VmSize\|VmHWM\|RSS\|Rss\|VmData\|procStat' status.log > trimmed.log
grep 'Process\|Cycle\|Date\|VmPeak\|VmSize\|VmHWM\|RSS\|Rss\|VmData' status.log > trimmed.log
#Create tempfile for processing
sed -i 's/Process:/Process: /g' trimmed.log
sed -i 's/Cycle:/Cycle: /g' trimmed.log
sed -i 's/Date:/Date: /g' trimmed.log
cp trimmed.log tempfile
cp trimmed.log tempfiler
chmod 750 tempfile tempfiler
# Replace semicolons with empty space
sed -i 's/:/ /g' tempfile
grep -Eo '^[^ ]+' tempfile > tempfile2

awk '!seen[$0]++' tempfile2 > tempfile3
#replace newline with comma for csv purposes
tr '\n' ',' < tempfile3 > output.log

#replace last comma with newline for csv purposes
#truncate -s-1 output.log
#sed -i -e '$a\' output
sed -i '$ s/.$/\n/' output.log

# take second column from trimmed data for csv purposes
gawk '{print $2}' tempfiler > tempfile

tr '\n' ',' < tempfile > experiment1.log
sed -i 's/,,/\n/g' experiment1.log

sed -i '$ s/.$/\n/' experiment1.log
#Change every 13 line to newline for csv purposes

perl -pe 's{,}{++$n % 13 ? $& : "\n"}ge' experiment1.log > experiment2.log
sed '1s/^.//' experiment2.log >> output.log

sed -i 's/,(Memtest.exe),/\n/g' output.log

#cleanup temporary files
rm tempfile tempfile2 tempfile3 tempfiler experiment1.log
```

Kuva 7 lokitiedot CSV formaatissa



```
Process,Cycle,#REF!,VmPeak,VmSize,VmHWM,VmRSS,RssAnon,RssFile,RssShmem,VmData,Date,0
andomnumbergenerationfor200000,0,0,6000,5964,3456,3456,128,3328,0,240,1717596975.422,1717596975422
Randomnumbergenerationfor200000,1,18,6000,5964,3456,3456,128,3328,0,240,1717596975.44,1717596975440
Randomnumbergenerationfor200000,2,38,6232,6168,3584,3584,256,3328,0,444,1717596975.46,1717596975460
Randomnumbergenerationfor200000,3,58,6428,6296,3712,3712,384,3328,0,572,1717596975.48,1717596975480
Randomnumbergenerationfor200000,4,78,6428,6296,3712,3712,384,3328,0,572,1717596975.5,1717596975500
Randomnumbergenerationfor200000,5,98,6812,6552,3968,3716,388,3328,0,828,1717596975.52,1717596975520
Randomnumbergenerationfor200000,6,118,6812,6552,3968,3716,388,3328,0,828,1717596975.54,1717596975540
Randomnumbergenerationfor200000,7,139,6812,6552,3968,3716,388,3328,0,828,1717596975.561,1717596975561
Randomnumbergenerationfor200000,8,159,7580,7064,4228,3828,500,3328,0,1340,1717596975.581,1717596975581
Randomnumbergenerationfor200000,9,179,7580,7064,4228,3828,500,3328,0,1340,1717596975.601,1717596975601
Randomnumbergenerationfor200000,10,198,7580,7064,4228,3828,500,3328,0,1340,1717596975.62,1717596975620
Randomnumbergenerationfor200000,11,219,7580,7064,4228,3828,500,3328,0,1340,1717596975.641,1717596975641
indexingRandomVectorForValues,0,0,7580,7064,4228,3956,628,3328,0,1340,1717596975.659,1717596975659
indexingRandomVectorForValues,1,20,7580,7064,4228,3956,628,3328,0,1340,1717596975.679,1717596975679
indexingRandomVectorForValues,2,38,7580,7064,4228,3956,628,3328,0,1340,1717596975.697,1717596975697
indexingRandomVectorForValues,3,57,7580,7064,4228,4084,756,3328,0,1340,1717596975.716,1717596975716
indexingRandomVectorForValues,4,76,7580,7064,4228,4212,884,3328,0,1340,1717596975.735,1717596975735
indexingRandomVectorForValues,5,96,7580,7064,4228,4212,884,3328,0,1340,1717596975.755,1717596975755
indexingRandomVectorForValues,6,115,7580,7064,4228,4212,884,3328,0,1340,1717596975.774,1717596975774
indexingRandomVectorForValues,7,133,7580,7064,4340,4340,1012,3328,0,1340,1717596975.792,1717596975792
indexingRandomVectorForValues,8,152,7580,7064,4340,4212,884,3328,0,1340,1717596975.811,1717596975811
indexingRandomVectorForValues,9,172,7580,7064,4340,4212,884,3328,0,1340,1717596975.831,1717596975831
indexingRandomVectorForValues,10,190,7580,7064,4340,4340,1012,3328,0,1340,1717596975.849,1717596975849
indexingRandomVectorForValues,11,210,7580,7064,4340,4212,884,3328,0,1340,1717596975.869,1717596975869
indexingRandomVectorForValues,12,230,7580,7064,4340,4212,884,3328,0,1340,1717596975.889,1717596975889
indexingRandomVectorForValues,13,249,7580,7064,4340,4212,884,3328,0,1340,1717596975.908,1717596975908
indexingRandomVectorForValues,14,268,7580,7064,4340,4212,884,3328,0,1340,1717596975.927,1717596975927
indexingRandomVectorForValues,15,287,7580,7064,4340,4084,756,3328,0,1340,1717596975.946,1717596975946
indexingRandomVectorForValues,16,307,7580,7064,4340,4212,884,3328,0,1340,1717596975.966,1717596975966
indexingRandomVectorForValues,17,324,7580,7064,4340,4212,884,3328,0,1340,1717596975.983,1717596975983
indexingRandomVectorForValues,18,344,7580,7064,4340,4212,884,3328,0,1340,1717596976.003,1717596976003
indexingRandomVectorForValues,19,362,7580,7064,4340,4212,884,3328,0,1340,1717596976.021,1717596976021
indexingRandomVectorForValues,20,382,7580,7064,4340,4084,756,3328,0,1340,1717596976.041,1717596976041
indexingRandomVectorForValues,21,400,7580,7064,4340,3956,628,3328,0,1340,1717596976.059,1717596976059
SortingRandomnumberstominmaxorderbystdsort,0,0,7884,7848,4724,4724,1396,3328,0,2124,1717596976.077,1717596976077
```

7.5 Pääohjelma

Pääohjelma kokoaa kaikki muut ohjelmat yhteen (kuva 8). Se Käyttää Bash kieltä ja aluksi se ajaa tietokoneen keräystiedot sen jälkeen se ajaa muistinkäyttöä kokeilevan ohjelman 20 kertaa että mahdolliset käyttöjärjestelmän muistin tarpeet eivät sotke tuloksia. Yhden kerran ajolla muiden ohjelmaien hetkellinen muistin tarve voi olla suuri ja tulokset voivat täten vääristyä. Seuraavaksi se trimmaa lokitiedot ja tuottaa CSV tiedoston. Koska muisti ohjelma on käännetty neljällä eri optimointi tasolla, ajetaan se niillä kaikilla läpi.

Kuva 8 Pääohjelman koodi

```
#!/usr/bin/bash
# This program is made for thesis work and can be used and modified freely.
# However any modifications can and probably will result in unwanted behavior. Use at own risk.
#
# This is only meant to be used in Linux environment.
# Creator: Rope Joonas
# 11.3.2024
#
# Note read README.txt before continuing!!!
# This program only combines several other small programs that are made for testing memory usage.
#
# First program collects hardware specifics and memory page size.
# Second Program Start C++ source code test.
# Third program trims logfile to more smaller and useable form.
# End of this program removes temp files if wanted.
# By default the temporary files are removed.
# These are not to be confused to temporary logfiles that have to be removed automatically by modifying this file.
# Last commands only need to be uncommented and it will be automatic
# For errors
set -e

# if the status.log doesn't exist

if [ ! -f "status.log" ]; then
        touch status.log
fi
# set status.log size to zero
truncate -s 0 status.log
./hardware.sh
sudo turbostat --Summary --quiet --show Busy%,Avg_MHz,PkgWatt --interval 0.001 --enable Time_Of_Day_Seconds -out turbo_test.log &
for((iii = 0; iii <= 20; iii++)) do
./Memtest.exe
done
# kill turbostat
sudo pkill turbostat
#echo -en Date: >> turbo_test.log
#chmod 755 turbo_test.log
#sudo date +'%s,%3N' >> turbo_test.log
./log_trimmer.sh
cp output.log output.csv
#rm status.log
```

8 Tulokset

Tulokseksi tuli CSV tiedosto, joka ajetaan Exceliin ja tuotetaan käyrät ajan, muistin käytön, keskusprosessorin aktiivisuuden, keskiarvo Megahertsien ja wattikulutuksen funktioina. Ohjelma ajetaan 20 kertaa ja sieltä valitaan kolme sykliä vertailtavaksi analyysin saa tehtyä yhdelläkin mutta loput ovat varmistusta varten.

Tuloksissa haittaa aiheuttaa myös jatkuva lokitietojen kirjoitus tiedostoon. Joka kerta kun muistinkäyttöä kokeilevaa ohjelmaa ajetaan niin se kirjoittaa status.log tiedostoon tietoa ja joka 1 - 5 ms välein turbostat ohjelma tekee saman omaan lokitiedostoonsa. Nämä aiheuttavat piikkejä keskusprosessorissa koska hitain ja kallein operaatio muistin kannalta on kovalevylle kirjoittaminen. Ohjelman ajaminen ilman lokitietoja taas ei kerro missä vaiheessa mikäkin kohta on ja kuinka paljon muistia on varattu. Mahdollisuus on ajaa ohjelma kahdeksalla eri versiolla eli kullakin optimointitasolla ja kerran täysillä lokikirjauksilla ja kerran vain eri osien alussa ja lopussa kirjoittamatta välivaiheita lokiin. Tämä toimii vain muistinkäyttöä seuraavan ohjelman osalta.

Erikseen energiankulutuksen laskemiseen olisi vielä pitänyt tietää mikä prosessoriydin kirjoittaa tietoa SWAP- osioon ja miksi. Työssä käytettävässä koneessa on 4 ydintä ja tasot L1, L2 ja L3. Vaikka tiedetään kuinka paljon on kilotavuina kirjoitettu niin pitäisi vielä tietää mitä prosessoriydintä Linux on käyttänyt kirjoitukseen.

8.1 Lenovo (Fedora Linux) Y50 -70 kannettavan tulokset

Koska prosessi ajettiin kaksikymmentä kertaa kestäen yhteensä 40 minuuttia otetaaan sykli 1 tarkasteltavaksi ja varmistetaan tietojen paikkansa pitävyys sykleistä 7 ja 15. Tällä yritetään varmistaa etteivät taustaohjelmat ole vaikuttaneet sykleihin liikaa.

Ohjelman rakenteesta ja tutkimistavasta johtuen alkuun ajetaan suuri määrä satunnaislukuja satunnasilukugeneraattorilla. Ilman satunnaisuutta kaikkia ohjelman kohtia ei voitaisi laskea. Satunnaisluvut olisi voinut ajaa ennen ohjelmaa tiedostoon mutta tiedoton avaaminen vaikuttaa keskusmuistin kuormitukseen joten parasta oli ajaa se ohjelmassa joka kerta erikseen.

Ohjelman tarkoitus oli tutkia keskusmuistin käyttöä ja energiasäästöä joten ensimmäinen testi oli erot eli lukujen järjestäminen numerojärjestykseen C++ algoritmillä ja perinteisellä tavalla. Numeroiden järjestäminen on huomattavan paljon käytetty ohjelmissa joten sen energiatehokkuuden paraneminen olisi merkittävä asia.

Kaikissa taulukoissa käytetään samoja arvoja jotta niitä voidaan vertailla paremmin. VMPeak,VmSize,VmRSS, RssAnon ja RssFile. Selitykset niille löytyvät liitteestä 4. Lyhyesti VMPeak virtuaalisen muistin korkein arvo ohjelman aikana. VmSize Virtuaalisen muistin koko. VmRss kertoo kuinka paljon virtuaalisesta muistista sijaitsee keskusmuistissa. RssAnon kertoo kuinka paljon muistista ei ole varmistettu tiedostolla. RssFile kertoo kuinka paljon muistista on oikeasti keskusmuistissa ja on varmistettu tiedostolla.

Taulukossa 1 on kuvattu erot muistinkäytön osalta Algoritmillä ja perinteisellä tavalla. Suurimmat erot ovat toki ajassa joka oli yli 500 kertainen ja vei koko ohjelman ajasta suurimman osan. Keskimäärin koko ohjelma kesti noin 48 - 49 sekuntia. VmRSS ja RssAnon tuli olla algoritmi tavalla nolla kuten syklissä 1, koska kesken std:sort komennon ei voi ottaa välikaappauksia lokitiedostoon.

Taulukko 1 Aineiston järjestely eri tavoilla

Algoritmi tapa Sykli	Aika ms	VMPeak kB	VmSize kB	VmRSS kB max -min	RssAnon kB max -min	RssFile kB
1	85	7892	7740	0	0	3480
7	86	7892	7856	128	128	3480
15	89	7892	7856	128	128	3476
Perinteinen tapa Sykli						
1	44556	7892	7740	256	256	3480
7	44478	7892	7744	384	384	3480
15	44514	7892	7744	256	256	3476

Toinen tarkastelu oli lukujen etsintä järjestetystä ja järjestämättömästä vektorista. Taulukon 2 mukaan tavalla ei näyttänyt olevan merkitystä kokeeseen. Syklissä 7 järjestämättömän tavan kohdalla oli RssAnon kohdalla piikki muistinkäytössä todennäköisesti syynä oli käyttöjärjestelmän muistin ylikäyttö.

Taulukko 2 Keskusmuistin käyttö lukujen järjestelystä

Järjestämätön tapa Sykli	Aika ms	VMPeak kB	VmSize kB	VmRSS kB max -min	RssAnon kB max -min	RssFile kB
1	316	7588	7072	384	384	3480
7	324	7588	7072	128	256	3480
15	317	7588	7072	384	384	3476
järjestetty tapa Sykli						
1	315	7892	7072	384	384	3480
7	322	7892	7856	256	256	3480
15	319	7892	7856	384	384	3476
Järjestämätön tapa lopilla Sykli						
1	308	7892	7740	128	128	3480
7	313	7892	7744	384	384	3480
15	308	7892	7744	256	256	3476
Järjestetty tapa ilman looppia Sykli						
1	304	7892	7740	256	256	3480
7	318	7892	7744	256	256	3480
15	312	7892	7744	384	384	3476
Järjestetty tapa loopilla Sykli						
1	314	7892	7740	256	256	3480
7	315	7892	7744	384	384	3480
15	314	7892	7744	256	256	3476

Taulukossa 3 ei ole näkyvissä lähes mitään eroa eri syklien tai ohjelmointitapojen välillä. Ajallisesti erot ovat mittausvirheen sisällä koska tätä ohjelmaa ei ole tarkoitettu ajettavaksi millisekunti tasolla.

Taulukko 3 laskentatapojen väliset erot keskusmuistin osalta

Tavallinen Pyramidi	Aika	VMPeak	VmSize	VmRSS	RssAnon	RssFile
Sykli	ms	kB	kB	kB	kB	kB
				max -min	max -min	
1	300	7892	7740	256	256	3480
7	299	7892	7744	384	384	3480
15	301	7892	7744	256	256	3476
Vaikea pyramidi						
Sykli						
1	300	7892	7740	128	128	3480
7	299	7892	7744	384	384	3480
15	300	7892	7744	256	256	3476
tavallinen ympyrä						
Sykli						
1	300	7892	7740	256	256	3480
7	300	7892	7744	256	256	3480
15	300	7892	7744	256	256	3476
vaikea ympyrä						
Sykli						
1	300	7892	7740	256	256	3480
7	299	7892	7744	384	384	3480
15	299	7892	7744	256	256	3476
tavallinen neliö						
Sykli						
1	299	7892	7740	256	256	3480
7	299	7892	7744	256	256	3480
15	299	7892	7744	256	256	3476
vaikea neliö						
Sykli						
1	299	7892	7740	256	256	3480
7	300	7892	7744	256	256	3480
15	299	7892	7744	256	256	3476

9 Yhteenveto

Koko työn tarkoituksena oli tutkia eri ohjelmointitapojen vaikutusta keskusmuistiin. Siinä tulokset jäivät laihoiksi. Vaikka tarkoituksena oli etsiä muistinkäytön osalta eroja algoritmien, perinteisen ja ohjelmien osittamisen avulla. Suurin ero oli käyttää std::sort komentoa perinteiseen tyyliin verrattuna jossa aikasäästö oli 500 kertainen. Mutta muistin käyttöä ei pystynyt vertailemaan koska algoritmista ei saa ulos välitietoja.

Järjestelyn yli 500 kertainen aikaero antaa kuitenkin mahdollisuuden pohtia kuinka paljon vähemmän aikaa se varaa keskusmuistia. Eli keskusmuistin täyttyessä Linux alkaa käyttämään swap osiota, jolloin kirjoitetaan osia keskusmuistista kovalevylle joka on hitaampaa ja vaatii lisää energiaa keskusprosessorin toimesta. Vaikka kokeessa käytetty testiohjelma on tarkoitettu käymään yksin niin oikessa tilanteessa taustaohjelmia ja muita prosesseja voi olla paljon yhtäaikaa päällä. Kaikilla niillä on tietty määrä keskusmuistissa tietoa tallennettuna. Täten mitä vähemmän aikaa keskusmuistia varataan sitä enemmän muut ohjelmat pääsevät käyttämään sitä ennenkuin tarvitaan swap- osiota.

Järjestelyn osalta tulokset vastasivat odotuksia eli algoritmi oli nopeampi kuin perinteinen tapa. Algoritmin osalta välituloksia oli mahdoton saada mutta perinteisen tavan osalta joka kierrokselta käytetty aika väheni odotetusti.

Erillaisten yksinkertaisten laskujen jakaminen omiin alaohjelmiin ei vaikuttanut tuloksiin käytännössä ollenkaan. Ajallisesti tuloksissa mennään virherajalla ja ohjelman muistin tutkimisosa joutui kokoajan pyörimään taustalla, joka selittää osin tuloksia. Jos keskusmuistissa ei ole ollut paljon varauksia niin ohjelman vaihe on voinut mennä nopeammin.

Yksinkertaisen laskujen osalta odotuksena oli että ohjelman kutsuessa ohjelmaa muistia käytettäisi enemmän tai aikaa kuluisi enemmän. Mutta tulosten valossa näin ei näyttänyt käyvän vaan mahdollisesti kääntäjä oli optimoinut jo käännösvaiheessa tai Linux ajon aikana.

Etsiminen järjestetyistä ja järjestämättömistä vektoreista ei myöskään näyttänyt vaikuttavan ohjelman käyntiaikaan. Järjestämätön tapa näytti olevan hitain keskiavolla kolmesta tarkasteltavasta syklistä mikä sopisi teoriaan mutta erot ovat millisekunteina niin pieniä että mittaustarkkuus alkaa olemaan ongelma.

Etsimisen osalta odotuksena oli että järjestämätön tapa olisi hitain, joka saattoi tapahtua. Mutta tavoissa ei näyttänyt olevan mittavia eroja.

Vaikka eroja ei tullut kovinkaan paljon paitsi järjestelytavassa niin monimutkaisemman ohjelman kanssa olisi tullut uusia syvyystasoja ongelman ratkaisuun. Suurimpia ongelmia oli että proc/PID tiedosto päivittyy vain silloin kun sieltä haetaan jotain ja sen jälkeen vanha tieto on poissa. Ja koska Linux määrittää prosessi ID:n vasta kun ohjelma käynnistyy ei sitä varten voi tehdä omaa seuranta ohjelmaa vaan se tulee aloittaa vasta sen sen jälkeen kun ohjelma on alkanut.

Toinen ongelma oli että Linux optimoi keskusmuistin käyttöä erittäin tehokkaasti. Vaikka numeroita luotiin miljoonia ja käyttettiin vanhempaa konetta, ei keskusmuisti täyttynyt tarpeeksi, että olisi päässyt kunnolla kokeilemaan Swap osiota joka on kaikkein hitain ja energiaa vievin vaihe.

Kolmas ongelma tuli GNU kääntäjästä, josta optimointia ei saanut pois päältä eli kääntäjä itsessään optimoi tehtyä ohjelmaa. Manuaalista löytyi risteäviä tietoja miten eri liput toimivat.

Koko ohjelman osalta olisi ollut parempi tehdä 20 pienempää ohjelmaa ja ajaa ne yksitellen peräkkäin ja ottaa kustakin omat tiedot ylös. Nyt viimeisten vaiheiden kohdalla oli keskusmuistiin jo syötettynä tietoa muista ohjelman osista. Varsinkin vektorit eivät automaattisesti nollaudu pois keskusmuistista ellei niitä erikseen aseteta nollaksi ja siitä huolimatta Linux saattaa pitää keskusmuistia varattuna nille varsinkin jos tilaa on jäljellä.

Turbostat ohjelman käyttö ei hyödyttänyt mitään, kun kyseessä oli keskusmuistin tutkiminen sen tulokset jäivät laihoiksi ja pääasiassa kaikki energiapiikit osuivat dokumentoivaan ohjelmaan. Eli ohjelmaan joka kirjoitti tietyin välein proc/PID tiedoston kovalevylle että sitä voitaisiin tutkia jälkikäteen. Lisäksi energian käytön laskemisen teki mahdottamaksi kun ei voitu tietää mikä prosessoriydin hoiti juuri tämän ohjelman kirjoittamisen SWAP- osiolle.

Lähteet

Computer Science Wiki. 3.2.2023.a. Heap memory. Haettu 1.3.2024. https://computersciencewiki.org/index.php?title=Heap_memory

Computer Science Wiki. 10.5.2023.b. Stack memory. Haettu 1.3.2024. https://computersciencewiki.org/index.php?title=Stack_memory

Computer Science Wiki. 9.10.2022. Haettu: 2.3.2024. https://computersciencewiki.org/index.php?title=Architecture_of_the_central_processing_unit_(CPU)

CPPReference. 13.11.2023.a. std::vector. Haettu: 4.3.2024. https://en.cppreference.com/w/cpp/container/vector

Crystal Rajeev.27.09.2023. Understanding Random Access Memory (RAM): A Crucial Component of Computing. American Journal of Computer Science and Engineering Survey. ISSN: 2349-7238. https://www.primescholars.com/articles/understanding-random-access-memory-ram-a-crucial-component-of-computing.pdf

Debian. 2024. Turbostat. Haettu 6.3.2024. https://manpages.debian.org/testing/linux-cpupower/turbostat.8.en.html

Gayathri, K, S. 05.2015. Memory management in C++ and java.https://www.researchgate.net/publication/275099106_MEMORY_MANAGEMENT_IN_C_AND_JAVA

Gary, S. 7.9.2007. All about Linux swap space. https://www.linux.com/news/all-about-linux-swap-space

Geeks for geeks. 20.12.2023.a. Difference between Static Arrays and Dynamic Arrays. https://www.geeksforgeeks.org/difference-between-static-arrays-and-dynamic-arrays/

GNU. 2024-a. GCC, the GNU Compiler Collection . Haettu 2.3.2024. https://gcc.gnu.org/

GNU. 2024-b. C++ Standards Support in GCC. Haettu: 2.3.2024. https://gcc.gnu.org/projects/cxx-status.html#cxx20

How-to-geek. 30.5.2023. L1, L2, and L3 Cache: What's the Difference?. https://www.howtogeek.com/891526/l1-vs-l2-vs-l3-cache

Kernel. (n.d). Concepts overview. Haettu 1.3.2024. https://docs.kernel.org/admin-guide/mm/concepts.html

Nwachukwu, C. 3.7.2023. C Programming: Mastering Flags in GCC. https://medium.com/@promisevector/c-programming-mastering-flags-in-gcc-32809491f340

Perez, J. L. 2024. Finding the Power Consumption of a Machine in Linux. Haettu 5.3.2024. https://www.baeldung.com/linux/power-consumption

Reynolds, L. 13.6.2021. ps output – Difference between VSZ vs RSS memory usage. https://linuxconfig.org/ps-output-difference-between-vsz-vs-rss-memory-usage

Vassev, E. Paquet, J. 06.2006. Aspects of Memory Management in Java and C++. https://www.researchgate.net/publication/221610499_Aspects_of_Memory_Management_in_Java_and_C

Wikipedia. 2024-a. Resident Set size. Haettu 9.3.2024. https://en.wikipedia.org/wiki/Resident_set_size

Wikipedia. 2024-b. Proportional set size. Haettu 9.3.2024. https://en.wikipedia.org/wiki/Proportional_set_size

Liite 1. Työssä tarvittavat kirjastot ja ohjelmat

Lshw

Getconf

Lscpu

Make

Gnu

G++

Build-essentials

Liite 2. Bash ohjelma tietokoneen tietojen keräys

```
#!/usr/bin/bash

# Author Rope Joonas 10.03.2024

#This program is designed to collect necessary information like hardware and operating system info.

#For the purpose of testing example program related memory and power usage for final thesis.

#Note lshw, getconf, lscpu have to be installed for this script to work

#lshw requires super user priviledges to operate this program my not work like intended.

#If you don't have those permissions

#This work doesn't require too much information.

#And since results are recorded IP addresses are omitted by -sanitize command

#hardware.info home folder.

PATH_TO_FILE=/home/$USER/Hardware_info

sudo lshw -sanitize -short > $PATH_TO_FILE/hardware.info



#Some systems require giving accees to file 750 or 755 are good enough.

#Since script needs superuser priviledges.

chmod 755 $PATH_TO_FILE/hardware.info

# We don't need input data for this research hence last lines are discarded
```

```
# and all lines that don't impact the test like mouse, bluetooth etc.

# Also due to case insensitivity practices flag -i must be on.

tail -n +3 $PATH_TO_FILE/hardware.info  | head -n -10 > tmp_file && mv tmp_file
$PATH_TO_FILE/hardware.info

sed -e "/camera/Id" $PATH_TO_FILE/hardware.info   > tmp_file && mv tmp_file
$PATH_TO_FILE/hardware.info

sed -e "/mouse/Id" $PATH_TO_FILE/hardware.info                > tmp_file && mv tmp_file
$PATH_TO_FILE/hardware.info

sed -e "/bluetooth/Id" $PATH_TO_FILE/hardware.info            > tmp_file && mv tmp_file
$PATH_TO_FILE/hardware.info

sed -e "/pnp/Id" $PATH_TO_FILE/hardware.info                  > tmp_file && mv tmp_file
$PATH_TO_FILE/hardware.info

sed -e "/ideapad/Id" $PATH_TO_FILE/hardware.info  > tmp_file && mv tmp_file
$PATH_TO_FILE/hardware.info

sed -e "/wireless/Id" $PATH_TO_FILE/hardware.info  > tmp_file && mv tmp_file
$PATH_TO_FILE/hardware.info

sed -e "/HDMI/Id" $PATH_TO_FILE/hardware.info                 > tmp_file && mv tmp_file
$PATH_TO_FILE/hardware.info

sed -e "/xhci/Id" $PATH_TO_FILE/hardware.info                 > tmp_file && mv tmp_file
$PATH_TO_FILE/hardware.info

sed -e "/hda/Id" $PATH_TO_FILE/hardware.info                  > tmp_file && mv tmp_file
$PATH_TO_FILE/hardware.info

sed -e "/ehci/Id" $PATH_TO_FILE/hardware.info                 > tmp_file && mv tmp_file
$PATH_TO_FILE/hardware.info
```

```
sed -e "/network/Id" $PATH_TO_FILE/hardware.info  > tmp_file && mv tmp_file
$PATH_TO_FILE/hardware.info

sed -e "/multimedia/Id" $PATH_TO_FILE/hardware.info                > tmp_file && mv tmp_file
$PATH_TO_FILE/hardware.info

sed -e "/PCI Express/Id" $PATH_TO_FILE/hardware.info    > tmp_file && mv tmp_file
$PATH_TO_FILE/hardware.info

sed -e "/hub/Id" $PATH_TO_FILE/hardware.info        > tmp_file && mv tmp_file
$PATH_TO_FILE/hardware.info

# Get rid of useless data like mounting points in list so far

awk '{$1=$2="";print$0}' $PATH_TO_FILE/hardware.info                > tmp_file && mv tmp_file
$PATH_TO_FILE/hardware.info

#Find out page size in operating system

printf "  Page Size: " >>  $PATH_TO_FILE/hardware.info

getconf PAGE_SIZE >> $PATH_TO_FILE/hardware.info

# Print CPU capacity and length needed for calculations later

printf "  CPU capacity and bus width:\n" >> $PATH_TO_FILE/hardware.info

sudo lshw -C cpu | grep -E 'width|capacity' >> $PATH_TO_FILE/hardware.info

# Print CPU bus address sizes and cache aligment (needed for page table calculations)

printf "  CPU bus address sizes and cache:\n" >> $PATH_TO_FILE/hardware.info

cat /proc/cpuinfo | grep -E 'address|cache' | head -n3 >> $PATH_TO_FILE/hardware.info

cat $PATH_TO_FILE/hardware.info
```

Liite 4. Status.log esimerkki yhdeltä sykliltä

Process:Randomnumbergenerationfor200000

Cycle:0

Date:05/03/24:17:03,13,142

Name:	Memtest.exe			
Umask:	0022			
State:	S (sleeping)			
Tgid:	3881			
Ngid:	0			
Pid:	3881			
PPid:	3820			
TracerPid:	0			
Uid:	1000	1000	1000	1000
Gid:	1000	1000	1000	1000
FDSize:	256			
Groups:	10 1000			
NStgid:	3881			
NSpid:	3881			
NSpgid:	3820			

NSsid: 2948

Kthread: 0

VmPeak: 5992 kB

VmSize: 5956 kB

VmLck: 0 kB

VmPin: 0 kB

VmHWM: 3584 kB

VmRSS: 3584 kB

RssAnon: 128 kB

RssFile: 3456 kB

RssShmem: 0 kB

VmData: 240 kB

VmStk: 132 kB

VmExe: 36 kB

VmLib: 3340 kB

VmPTE: 52 kB

VmSwap: 0 kB

HugetlbPages: 0 kB

CoreDumping: 0

THP_enabled: 1

untag_mask: 0xffffffffffffffff

Threads: 1

SigQ: 0/31050

SigPnd: 0000000000000000

ShdPnd: 0000000000000000

SigBlk: 0000000000010000

SigIgn: 0000000000000006

SigCgt: 0000000000000000

CapInh: 0000000800000000

CapPrm: 0000000000000000

CapEff: 0000000000000000

CapBnd: 000001ffffffffff

CapAmb: 0000000000000000

NoNewPrivs: 0

Seccomp: 0

Seccomp_filters: 0

Speculation_Store_Bypass: thread vulnerable

SpeculationIndirectBranch: conditional enabled

Cpus_allowed: ff

Cpus_allowed_list: 0-7

Mems_allowed:

00000000,00000001

Mems_allowed_list: 0

voluntary_ctxt_switches: 16

nonvoluntary_ctxt_switches: 0

x86_Thread_features:

x86_Thread_features_locked:

1489 896 864 9 0 93 0

procStat:3881 (Memtest.exe) S 3820 3820 2948 34816 3820 4194304 158 2087 0 0 0 0 0 320 0 1 0 397873 6098944 896 18446744073709551615 4202496 4238337 140731057868576 0 0 0 65536 6 0 1 0 0 17 6 0 0 0 0 0 4259264 4260316 10936320 140731057873406 140731057873420 140731057873420 140731057876970 0

Liite 5. Käsitesanasto

DMA (Direct Memory access). Laitteiston ohjelmat voivat käyttää keskusmuistia suoraan ilman keskusprosessoria.

LFS Linux. (Linux from scratch) Tässä kyseisessä versiossa Linux rakennetaan itse jo valmiina olevaan unix käyttöjärjestelmään. Optimoinnin kannalta tehokkain varsinkin erikoistarpeita silmällä pitäen ja erittäin kevytrakenteinen.

G++. GNU kääntäjä C++ ohjelmille.

Jaettu kirjasto. Kirjasto, jonka usea ohjelma jakaa yhtä aikaa keskusmuistista. (https://amir.rachum.com/shared-libraries)

Kirjasto (library). Sisältää jo käännettyä koodia ja tietoa.

Kernel. Kernel toimii siltana keskusprosessorin, RAM, laitteiden ja applikaatioiden välillä. Käytännössä käyttöjärjestelmä.

Käynninaikainen (Run- time). Käynninaikaisella toteutuksella tarkoitetaan ohjelmoinnissa ohjelmaa, joka voidaan ajaa mutta ei muuttaa.

Kääntämisen aikainen. (Compile time). Ohjelma käännetään ohjelmointikieleltä konekielelle.

RAM (Random access memory). Keskusmuisti. Hoitaa nopeasti tarvittavan muistin käyttöjärjestelmässä kuten keskusprosessorin suorittaman ohjelman muuttujat.

RssAnon kertoo kuinka paljon keskusmuistissa on tietoa jota ei ole varmistettu muualle

RssFile kertoo kuinka paljon ohjelma käyttää sivuina muistia.

SWAP osio. Osio antaa lisää mahdollisuuksia käyttöjärjestelmälle allokoida muistia, jos keskusmuisti on täynnä vaihtamalla muistipaikkoja vähemmän kriittisen muistin kanssa väliaikaisesti. Hitaampi kuin RAM.

Virtuaalinen muisti sisältää fyysisen keskusmuistin ja Swap osion

VMPeak kertoo suurimman käytetym uistin määrän ohjelman käynnin aikana.

VMSize kertoo ohjleman käyttämän koko virtuaalisen musitin määrän

vmRSS kertoo kuinka paljon ohjelma käyttää fyysistä keskusmuistia sillä hetkellä.

Liite 6. Tietokoneen kartoitusohjelman tiedot

Kannettava	Pöytätietokone
WS (SKU)	(LENOVO_MT_20378_BU_idea_FM_Lenovo Y50-70)
Z9PE-D8 WS	Lenovo Y50-70
64KiB BIOS	128KiB BIOS
Intel(R) Xeon(R) CPU E5-2630 0 @ 2.30GHz	Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz
384KiB L1 cache	32KiB L1 cache
1536KiB L2 cache	256KiB L2 cache
15MiB L3 cache	6MiB L3 cache
(To Be Filled By O.E.M.)	32KiB L1 cache
12GiB System Memory	8GiB System Memory
DIMM Synchronous [empty]	8GiB SODIMM DDR3 Synchronous 1600 MHz (0.6 ns)
4GiB DIMM DDR3 1333 MHz (0,8 ns)	DIMM [empty]
4GiB DIMM DDR3 1333 MHz (0,8 ns)	DIMM [empty]
4GiB DIMM DDR3 1333 MHz (0,8 ns)	DIMM [empty]
DIMM Synchronous [empty]	Xeon E3-1200 v3/4th Gen Core Processor DRAM Controller
DIMM Synchronous [empty]	GM107M [GeForce GTX 860M]
DIMM Synchronous [empty]	4th Gen Core Processor Integrated Graphics Controller
DIMM Synchronous [empty]	8 Series/C220 Series Chipset Family MEI Controller #1
Xeon E5/Core i7 DMI2	HM86 Express LPC Controller
GK110 [GeForce GTX TITAN]	storage 8 Series/C220 Series Chipset Family 6-port SATA Controller 1 [AHCI mode]
Xeon E5/Core i7 Address Map, VTd_Misc, System Management	disk 1TB WDC WD10SPCX-24H
Xeon E5/Core i7 Control Status and Global Errors	599MiB Windows FAT volume
Xeon E5/Core i7 I/O APIC	volume 1GiB EXT4 volume
C602 chipset 4-Port SATA Storage Control Unit	volume 929GiB EFI partition
C600/X79 series chipset MEI Controller #1	8 Series/C220 Series Chipset Family SMBus Controller
C600/X79 series chipset MEI Controller #2	CRB Battery 0
C600/X79 series chipset USB2 Enhanced Host Controller #2	Page Size: 4096
ASM1042 SuperSpeed USB Host Controller	CPU capacity and bus width:
ASM1042 SuperSpeed USB Host Controller	capacity: 3500MHz
storage 88SE9230 PCIe 2.0 x2 4-port SATA 6 Gb/s RAID Controller	width: 64 bits
disk 3TB WDC WD30EZRX-00D	CPU bus address sizes and cache:
volume 1023KiB BIOS Boot partition	cache size: 6144 KB
512MiB Windows FAT volume	cache_alignment: 64
volume 2794GiB EXT4 volume	address sizes : 39 bits physical, 48 bits virtual
C600/X79 series chipset USB2 Enhanced Host Controller #1	
volume 991MiB Windows FAT volume	
C600/X79 series chipset LPC Controller	
storage C600/X79 series chipset 6-Port SATA AHCI Controller	
C600/X79 series chipset SMBus Host Controller	
Page Size: 4096	
CPU capacity and bus width:	
capacity: 4GHz	
width: 64 bits	
CPU bus address sizes and cache:	
cache size: 15360 KB	
cache_alignment: 64	
address sizes: 46 bits physical, 48 bits virtual	