OULUN AMMATTIKORKEAKOULU

Samuli Linna

EVENT MACHINEN PAIKALLINEN TIEDONSIIRTO JAETUN MUISTIN AVULLA

EVENT MACHINEN PAIKALLINEN TIEDONSIIRTO JAETUN MUISTIN AVULLA

Samuli Linna
Opinnäytetyö
Kevät 2021
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehityksen suuntautumisvaihtoehto

---

Tekijä: Samuli Linna
Opinnäytetyön nimi suomeksi: Event Machinen paikallinen tiedonsiirto jaetun muistin avulla
Opinnäytetyön nimi englanniksi: Event Machine’s local data transmission with shared memory
Työn ohjaaja: Kari Jyrkkä
Työn valmistumislukukausi ja -vuosi: Kevät 2021
Sivumäärä: 29

Työn tilaaja Nokia Solutions and Network Oy haluaa tutkia tukiasemissa käytettävän Baseband Platform Services -ohjelmiston paikallisen tiedonsiirron parantamista. Opinnäytetyön tavoitteena on tutkia ja toteuttaa jaettuun muistiin perustuva prosessien välinen kommunikaatiomekanismi ja suorittaa vertailukelpoiset suorituskykymittaukset.

Opinnäytetyö aloitettiin tutustumalla mekanismin mahdolliseen toteutustapaan alustavien piirrosten perusteella, ja tutustuttiin Event Machine -alijärjestelmän toimintaan ja lähdekoodeihin. Nykyisen tiedonsiirtomekanismin ymmärtäminen oli tärkeää mekanismin suunnittelussa ja toteutuksessa.

Työtä tehtiin itsenäisesti toimivan Event Machine -alijärjestelmän prototyypin päälle, että saataisiin mahdollisimman vakaa ohjelmointiympäristö, jossa suuremmat alijärjestelmien ohjelmistomuutokset eivät vaikuttaisi huomattavasti opinnäytetyön etenemiseen.

Mekanismin prototyyppi saatiin toteutettua ja se saatiin integroitua pääjärjestelmään testien suorittamiseksi. Vertailukelpoiset suorituskykymittaukset saatiin tehtyä ja mittausten perusteella todettiin, että mekanismia ei sellaisenaan voida ottaa käyttöön tämän hetken tuotteissa.

Työn lopuksi arvioitiin, että mekanismia voitaisiin mahdollisesti käyttää tulevissa tuotteissa, joissa Event Machine pystyy hyödyntämään suurempia jaetun muistin sivuja.

Asiasanat: Event Machine, jaettu muisti, mäppäys, suorituskykytestaus

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, option of Software Development

---

Author: Samuli Linna
Title of thesis: Event Machine's local data transmission with shared memory
Supervisor: Kari Jyrkkä
Term and year when the thesis was submitted: Spring 2021
Pages: 29

Nokia Solutions and Networks Oy wants to examine a mechanism for Inter-Process Communication in base station software that uses shared memory. The goal of the work is to research and create a communication mechanism inside a base station that utilizes shared memory in Inter-Process Communication and run comparable performance tests.

Thesis was started by getting used to the programming environment and initial drawings of a possible implementation of IPC-mechanism. Event Machine's source codes were examined to get familiar with the programming environment and the current data transmission mechanism.

The work was done on top of an independently running Event Machine prototype to have a stable programming environment that is unlikely affected by greater changes in some linked subsystems.

Data transmission mechanism protype was finished and integration to main system was tested. Comparable performance tests were done and results showed that the mechanism couldn't be integrated as it stands. At the end of the thesis the conclusion was that the mechanism could be used in up coming products where the mechanism could utilize bigger shared memory pages.

Keywords: Event Machine, mapping, performance testing, shared memory

ALKULAUSE

Työ tehtiin Nokia Solutions and Networks Oy:lle. Haluan kiittää linjamanageri Janne Liimataista mahdollisuudesta suorittaa opinnäytetyö joustavasti osana perehdyttämistä uusiin työtehtäviin. Kiitokset opinnäytetyön ohjaajalle Kari Jyrkälle rakentavasta palautteesta ja ohjauksesta. Paljon kiitoksia ohjelmistoarkkitehti Teemu Aholalle opinnäytetyön aiheen ideoimisesta ja ohjauksesta. Kiitokset myös Tony Hakalalle ja Markus Jussilalle tärkeästä tuesta opinnäytetyön aikana.

Oulussa 11.6.2021

Samuli Linna

SISÄLLYS

SANASTO

BIP	BTS Intranet Protocol, protokolla tukiasemien sisäiseen tiedonsiirtoon.
EM	Event Machine, moniydinoptimoitu prosessointikonsepti.
Eventti	Eventti eli tapahtuma. Sovelluskohtaista dataa, kuten viestejä tai verkkopaketteja.
Hugepages	Linuxin muistisivuja, jotka ovat suurempia kuin 4 kilobittiä.
IPC	Inter-Process Communication, prosessien välinen kommunikaatio.
Overhead	Tuhlattu suoritusaika.
UP HWAPI	User-Plane Hardware Application Programming Interface, tukiaseman järjestelmäkomponentti.

1 JOHDANTO

Työn tilaajan Nokia Solutions and Network Oy haluaa tutkia tukiasemissa käytettävän Baseband Platform Services -ohjelmiston paikallisen tiedonsiirron suorituskyvyn parantamista. Opinnäytetyön tavoitteena on tutkia ja toteuttaa jaettuun muistiin perustuva prosessien välinen kommunikaatiomekanismi. Opinnäytetyön kommunikaatiomekanismi on yksi mahdollisista vaihtoehdoista, joka voitaisiin ottaa käyttöön Nokian tulevissa tuotteissa.

Nykyinen prosessien välinen paikallinen datansiirto tapahtuu protokollalla, jossa dataa kopioidaan, pilkotaan, siirretään ja vastaanotetaan tarpeettomasti. Työn aihe rajataan jaettuun muistiin perustuvaan kommunikaatiomekanismiin, jolla pyritään parantamaan datansiirtonopeutta ja pienentämään viivettä.

Aihetta valitessa koettiin tärkeäksi, että työ olisi mahdollisimman vähän riippuvainen suuremmista ohjelmistomuutoksista, jotka voisivat estää työn etenemisen. Tämän vuoksi kehitysympäristöksi valittiin ohjelmistokomponentin itsenäinen versio.

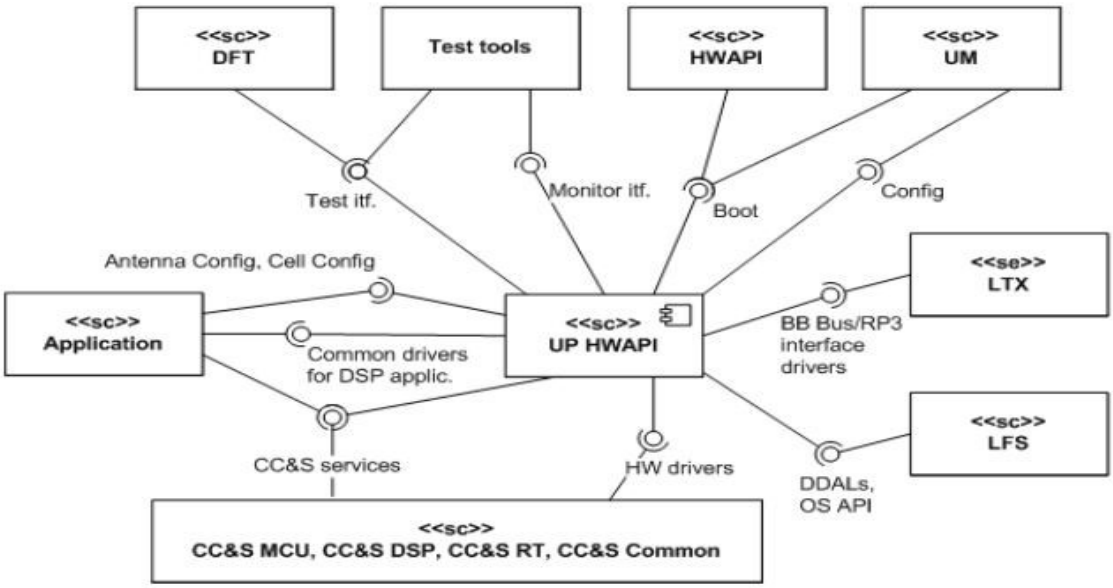
Lopuksi suoritetaan vanhan ja uuden mekanismin väliset vertailukelpoiset suorituskykymittaukset. Tiedonsiirtomekanismi integroidaan ohjelmiston tuotekehityshaaraan suorituskykymittauksia varten. Tuloksien perusteella päätetään jatkokehityksestä.

2 KÄSITTEIDEN JA OHJELMISTON ESITTELY

Tässä luvussa esitellään opinnäytetyössä käytettävät käsitteet ja niiden keskeinen toiminta järjestelmän tiedonsiirrossa.

2.1 UP HWAPI

User-Plane Hardware Application Programming Interface (UP HWAPI) on tukiasemien BB PS -ohjelmistoissa käytettävä järjestelmäkomponentti, joka piilottaa laitteiston ylemmän tason sovelluksilta ja toteuttaa tarvittavat ohjelmointirajapinnat (kuva 1). UP HWAPI tarjoaa joukon palvelualustoja suorituskykyä vaativille aikakriittisille käyttäjätason ohjelmistoille. (1.)



KUVA 1. UP HWAPI:n yleisnäkymä (1.)

2.2 Event Machine -alijärjestelmä

Event Machine on UP HWAPI:n alijärjestelmä, joka tarjoaa Event Machine -prosessointikonseptin toteutuksen ja rajapinnat OpenEM-viitekehyksen alustamiselle, halinnalle ja suorittamiselle. EM-alijärjestelmä löytyy kuvasta 1 UP HWAPI -komponentin sisältä.

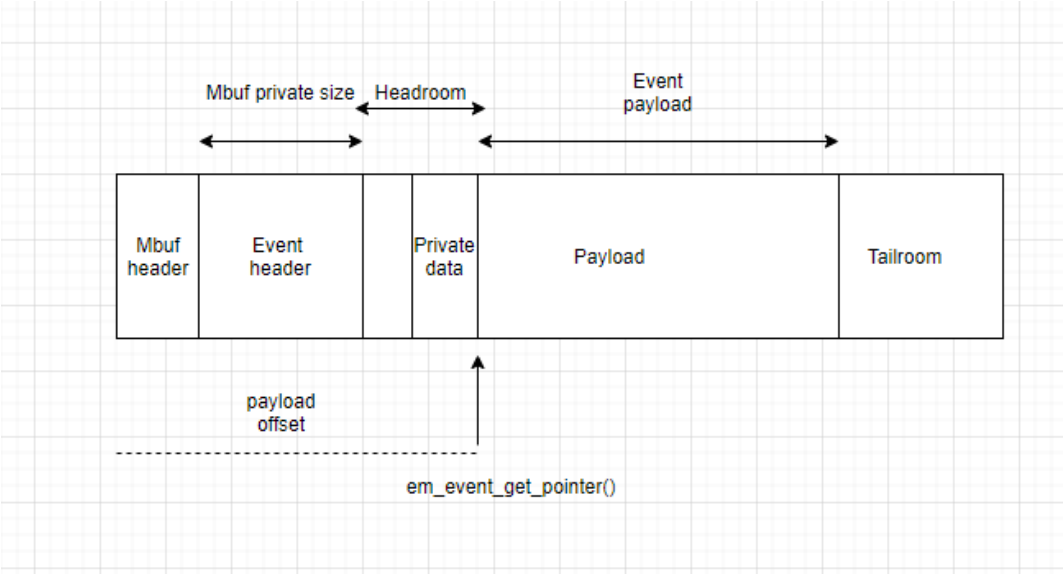
2.3 Event Machine -prosessointikonsepti

OpenEM eli EM on Nokia Siemens Networksin kehittämä korkean suorituskyvyn ajonaikainen dynaamisesti kuormitusta tasaava moniydinoptimoitu prosessointikonsepti. EM:n toiminta perustuu tapahtumien eli eventtien rinnakkaiseen käsittelyyn asynkronisten jonojen avulla. (2.)

2.3.1 Eventit

Eventit eli tapahtumat kuvaavat työtä, joka täytyy tehdä. Eventit ovat sovelluskohtaista dataa, kuten viestejä tai verkkopaketteja. Eventin tyyppi määrittelee sen rakenteen sovellusten käyttötarkoitusten mukaan. Kaikki EM-ympäristössä suoritettava prosessointi täytyy käynnistyä tapahtuman avulla. (3.)

Kuvan 2 mukaisia tapahtumia lähetetään EM:n jonoihin käsiteltäviksi. Tapahtuman datan ei tarvitse olla payloadissa, vaan kuvan 2 em_event_get_pointer()-funktiolla saadaan osoitin käsiteltävään dataan.



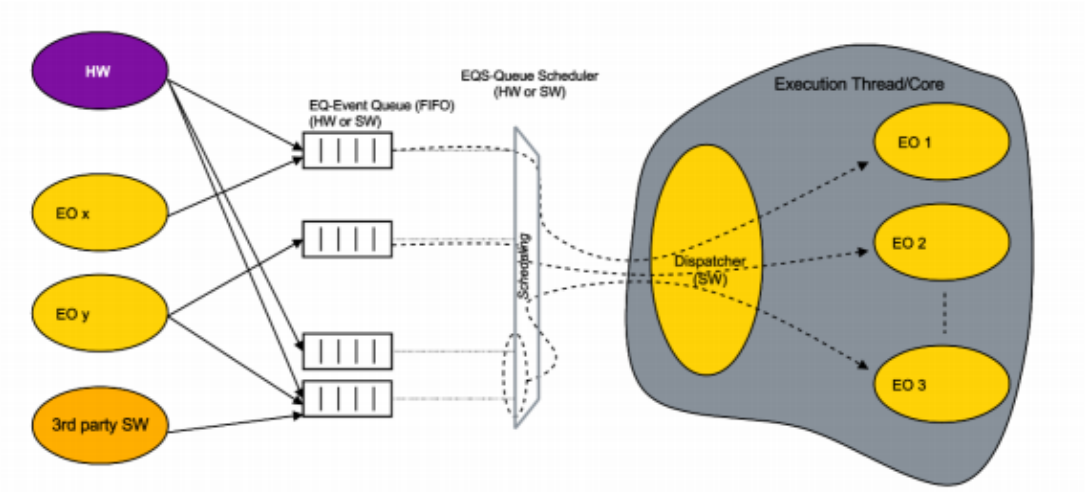
KUVA 2. Esimerkki tapahtuman rakenteesta

2.3.2 Tapahtumien prosessointi

EM:n päälle rakennetut sovellukset rakennetaan suoritusobjekteista (EO), jotka ovat yhteydessä tapahtumajonoihin.

EM:n päälle voidaan rakentaa sovelluksia, jotka voivat lähettää ja vastaanottaa tapahtumia. Sovellukset koostuvat suoritusobjekteista, joilla on start()-, receive()- ja stop()-funktiot. Näitä funkioita kutsumalla omassa prosessissaan ajettava käsittelijä hallinnoi tapahtumien kulkua. (Kuva 4.)

Laitteet ja kolmannen osapuolen sovellukset voivat myös lähettää tapahtumia käsiteltäviksi EM:n tapahtumajonoihin.



KUVA 3. Suoritusobjektit (EO), tapahtumajonot, tapahtumajärjestelijä ja käsittelijä (3.)

2.3.3 Tapahtumajonot

Tapahtumien lähettäjä valitsee haluamansa palvelun ja prioriteetin mukaan jonon, minne tapahtuma lähetetään. Suoritusobjektien ja jonojen välillä on yhteys siten, että jokaisella jonolla on oma suoritusobjekti, mutta suoritusobjekteilla voi olla yhteys moneen jonoon. Tapahtumajonoilla annetaan tyyppi, jonka määrittelee tapahtumien käsittelytavan.

Atomiset jonot

Atomisilla jonoilla voi olla vain yksi tapahtuma samaan aikaan. Näiden jonojen tehtävänä on ylläpitää tapahtumien järjestystä ja jonon kontekstidatan synkronointia. Atomiset jonot ovat tärkeitä sovelluksen skaalautuvuudelle.

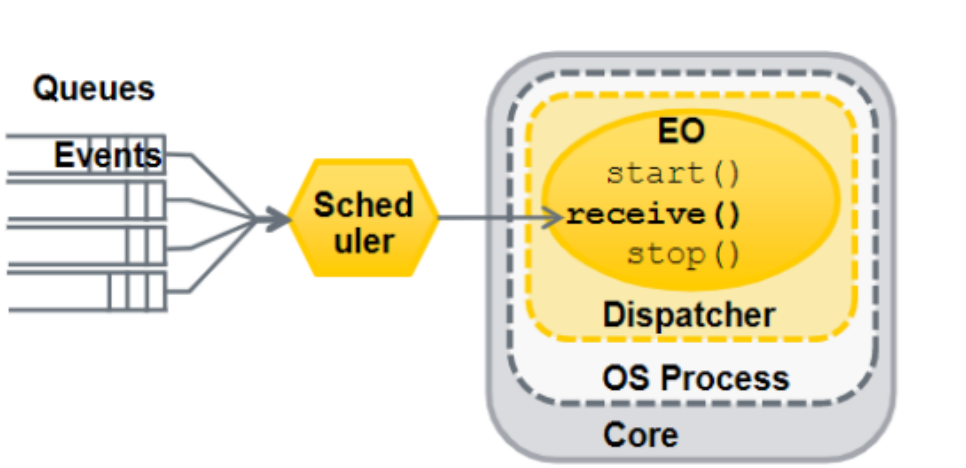
Rinnakkaiset jonot

Rinnakkaiset jonot mahdollistavat tapahtumien rinnakkaisen suorittamisen usealla ytimellä samanaikaisesti.

2.3.4 Käsittelijä ja tapahtumajärjestelijä

Käsittelijä on jokaisessa EM:lle varatussa ytimessä ajettava ohjelma, jota suoritetaan omassa prosessissaan. Käsittelijä pyytää tapahtumajärjestelijältä tapahtuman, joka tarkistaa jonojen tilan ja lähettää korkeimman prioriteetin tapahtuman käsittelijälle. (Kuva 3.)

Lopuksi käsittelijä ohjaa tapahtuman eteenpäin oikeaan jonoon kutsumalla suoritusobjektin receive()-funktiota. Kun suoritusobjektin receive()-funktio on suoritettu ja funktiosta palataan, voidaan seuraava tapahtuma vastaanottaa tapahtumajärjestelijältä. (Kuva 3.)



KUVA 4. Lähettäjä, jonot ja tapahtumakäsittelijä (3.)

2.4 Prosessien välinen kommunikaatio

2.4.1 BTS Intranet Protocol

BTS Intranet Protocol eli BIP on Nokian kehittämä vähäisen latenssin protokolla tukiasemien sisäiseen kommunikointiin. BIP vastaa tukiaseman sisällä tapahtuvasta kommunikoinnista. Tiedonsiirto BIP:n kautta tapahtuu Ethernetin yli. Opinnäytetyössä tehtävän tiedonsiirtomekanismin on tarkoitus korvata BIP EM:n välisessä tiedonsiirrossa.

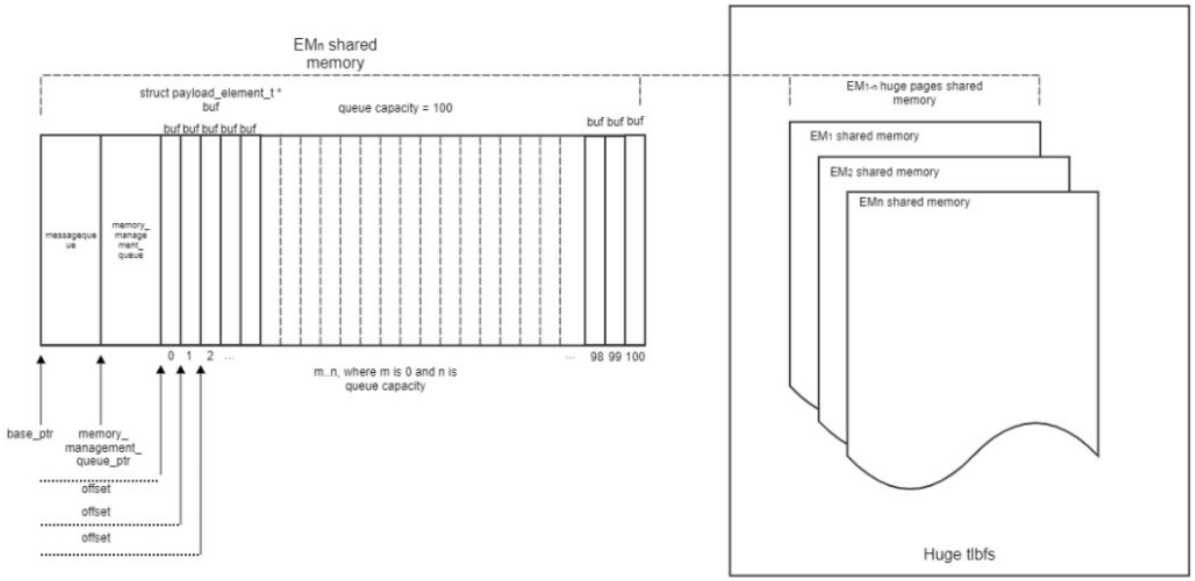
2.4.2 Prosessien välinen kommunikaatio jaetun muistin avulla

Jaettu muisti on nopein prosessien välinen kommunikointimenetelmä, jossa käyttöjärjestelmä päästää käyttämään muiden prosessien muistiavaruuksia. Prosessit voivat kirjoittaa ja lukea näille muistialueille ilman käyttöjärjestelmän systeemikutsuja. (4.)

2.5 Hugepages- jaettu muisti

Hugepages on Linuxin kerneliin integroitu ominaisuus, joka antaa käyttöjärjestelmälle mahdollisuuden tukea oletutuskokoa (4 KB) suurempia muistisivuja. Useimmat nykyiset suorittimet tukevat suurempia sivukokoja.

Hugepagejen hyödyllisyys tulee siitä, että käyttöjärjestelmän täytyy käydä läpi ja päivittää vähemmän muistialueita. Opinnäytetyön tiedonsiirtomekanismi käyttää 2 MB:n hugepageja. (5.)



KUVA 5. EM-instanssin hugepages- jaettu muisti ja sen sisältö

3 EM-PROSESSIEN VÄLINEN TIEDONSIIRTO

Opinnäytetyötä alettiin suunnittelemaan oletuksella, että EM:n välistä tiedonsiirtoa voidaan tehostaa korvaamalla BIP jollain vaihtoehtoisella IPC-mekanismilla. Opinnäytetyö keskittyi jaetun muistin avulla kehitettävään tiedonsiirtomekanismiin. Seuraavaksi esitellään opinnäytetyössä käsiteltävän uuden tiedonsiirtomekanismin toimintaperiaate.

3.1 Toimintaperiaate

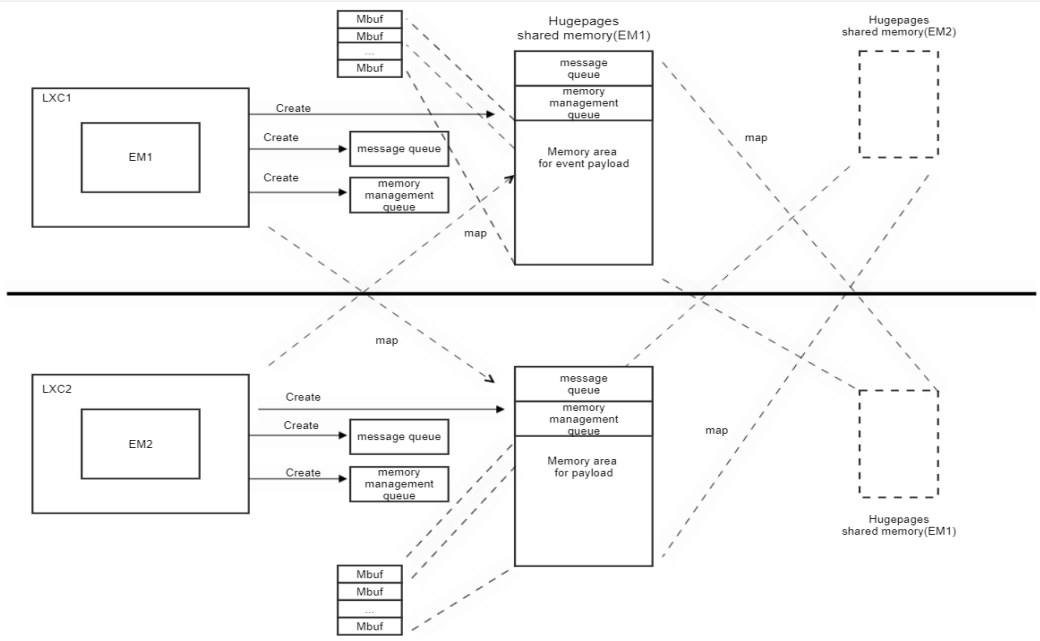
Jokainen EM saa käynnistyessään laitekonfiguraatiosta viereisten EM-instanssien laitetunnisteen. Laitetunniste on yksilöllinen ja se toimii nimenä jaetulle muistille.

Jokainen EM-instanssi luo itselleen hugepages- jaetun muistin ja mäppää (6) eli ottaa muistialueen käyttöönsä osaksi omaa muistiavaruutta. Nämä muistialueet on varattu pelkästään tiedonsiirtomekanismia varten. Jokainen instanssi myös mäppää viereisen instanssin jaetun muistin. Kuvassa 6 molemmat EM-instanssit suorittavat tämän toimenpiteen (”map”).

Jaetuissa muistialueissa varataan muistia jonoille ja viestien tietosisällöille (kuva 5). Tämän muistialueen koko määräytyy tuotekohtaisesti. Tässä opinnäytetyössä käytetään kahden megatavun kokoisia hugepageja.

Säiliöt

Säiliöt ovat ohjelmistopaketteja, jotka sisältävät kaiken tarpeellisen ohjelmiston suorittamiseen ympäristöstä riippumatta. Säiliöt virtualisoivat käyttöjärjestelmän resurssit ja mahdollistavat eristetyn ympäristön sovellusten kehittämiselle. (7.)



KUVA 6. Ylätason näkymä jaettuun muistiin perustuvasta tiedonsiirtomekanismista

3.2 Mekanismin alustaminen

Seuraavaksi käydään läpi opinnäytetyön tiedonsiirtomekanismin alustamisessa suoritettavat funktiot ja niiden toiminnot.

3.2.1 Oman jaetun muistin alustaminen

EM:n käynnistysvaiheessa suoritetaan siirtomekanismin tarvittavat alustusfunktiot. Työssä käytettävä siirtomekanismi alustetaan suorittamalla init_turbo_channel()-funktio (kuva 7). Funktio luo kuvan 6 Huge pages shared memory (EM1 / EM2)- jaetut muistit ja niiden sisälle tulevat viesti- ja muistinhallintajonot sekä varaa muistialueen eventtien payloadille.

```
DLL_PUBLIC void* init_turbo_channel(uint16_t device_id)
{
    void* turbo_channel_shm_ptr;
    turbo_channel_shm_ptr = create_and_map_turbo_channel_shm(device_id);
    create_and_init_queues(turbo_channel_shm_ptr, QUEUE_CAPACITY, BLOCK_SIZE);

    return turbo_channel_shm_ptr;
}
```

KUVA 7. Siirtomekanismin alustusfunktio lähdekoodissa

Alustusfunktiossa luodaan ja mäpätään hugepages-jaettu muisti kutsumalla create_and_map_turbo_channel_shm()-funktiota. Tälle funktiolle annetaan parametrina oma yksilöllinen laitetunniste, joka saadaan ohjelmiston käynnistysvaiheessa. Lopuksi funktio palauttaa osoittimen jaetun muistin alkuun, joka on kuvassa 5 base_ptr.

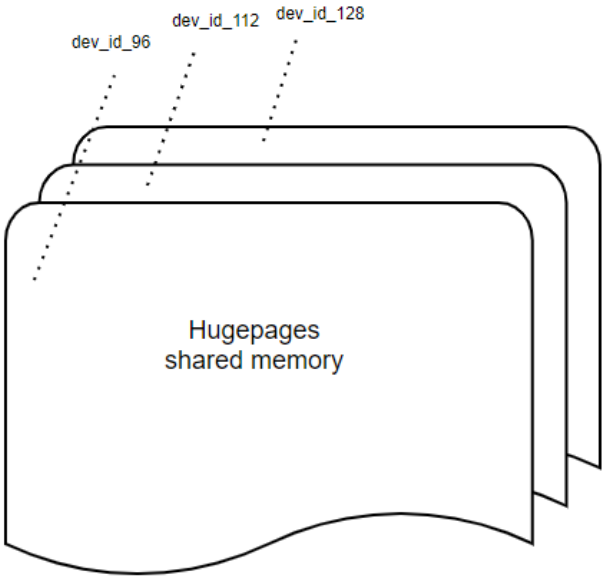
Seuraavaksi suoritetaan create_and_init_queues-funktio, jossa luodaan viesti- ja muistinhallintajonot (kuva 5). Tämän jälkeen varataan pienempiä muistialueita puskuriksi viestinlähetysjonon tietosisällölle. Näiden muistialueiden koko määritellään käsiteltävin tapahtumien koon mukaan. Muistialueiden koko täytyy aina olla suurempi kuin käsiteltävän tapahtuman koko.

Lopuksi alustusfunktio palauttaa osoittimen mäpätyn jaetun muistiin alkuun. Tämä osoitin on kuvan 5 base_ptr. Osoitin tallennetaan EM:n paikalliseen muistiin myöhempää käyttöä varten.

3.2.2 Viereisten EM-instanssien muistien alustaminen

Tiedonsiirtomekanismin toiminnan edellytyksenä on, että tapahtuman lähetettäessä jokaisella EM-instanssilla on tiedossa toistensa laitetunniste ja osoitin jaetun muistin alkuun.

Jokaisen EM-instanssin hugepaget on nimetty niiden laitetunnisteen mukaan. Nämä hugepaget ovat kaikkien instanssien nähtävissä tiedoistoina Linuxin juuren alahakemistossa /mnt/huge-2M. (Kuva 8.)



KUVA 8. Hugepages-jaetut muistit. Tiedostot on nimetty laitetunnisteen mukaan.

Kun jokainen EM-instanssi on luonut oman jaetun muistinsa, voidaan mäpätä viereisten instanssien jaetut muistit niiden laitetunnisteen avulla. Suoritetaan add_turbo_channel_neighbour_device_id-funktio jokaiselle EM-instanssille (kuva 9). Funktio tallettaa instanssien laitetunnisteen ja jaetun muistin osoittimen tietuetaulukkoon (kuva 10).

```
/*  Store neighbour node's deviceID and pointer to shared memory  */
bool add_turbo_channel_neighbour_device_id(uint16_t device_id)
{
    for(int i = 0; i < MAX_NODES; i++)
    {
        if(em_shm->s_neighbours[i].device.id == 0 && s_neighbours[i].shared_mem_ptr == NULL)
        {
            void* shm_ptr;
            shm_ptr = map_turbo_channel_shm(device_id);
            em_shm->s_neighbours[i].device_id = device_id;
            em_shm->s_neighbours[i].shared_mem_ptr = shm_ptr;

            return true;
        }
    }
    return false;
}
```

KUVA 9. EM-instanssien laitetunnisteiden ja jaetun muistin osoittimen tallentaminen tietuetaulukkoon

```
struct neighbour_node_t {
    uint16_t device_id;
    void* shared_mem_ptr;
};

neighbour_node_t s_neighbours[MAX_NODES];
```

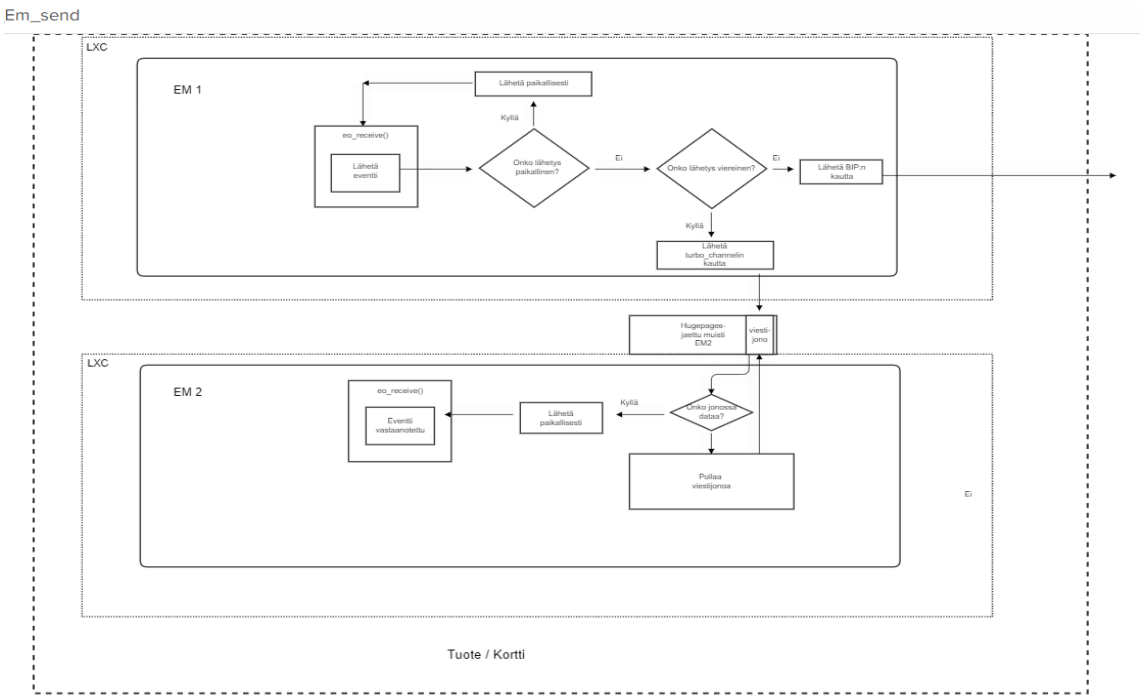
KUVA 10. Tietue ja tietuetaulukko

3.2.3 Säiliöiden ulkopuolisten lähetysten rekisteröinti

Käytössä oleva BIP-protokolla korvataan register_turbo_channel_tx_handler()-funktiolla. Funktio rekisteröi takaisinkutsufunktion, joka aktivoituu, kun tapahtumia lähetetään toiselle EM-instanssille.

3.3 Tapahtuman lähetys ja vastaanotto

Kuvassa 11 käydään läpi tapahtuman lähetyksen ja vastaanoton vaiheet kahden EM-instanssin välillä.



KUVA 11. Tapahtuman lähettäminen kahden EM:n välillä

Tapahtuman lähettäminen

Käsittelijä kutsuu suoritusobjektin eo_receive()-funktiota, joka käynnistää tapahtuman käsittelyn. Funktiossa lähetetään tapahtuma toiselle EM-instanssille, joka sijaitsee omassa säiliössään. Lähettäjä antaa em_send()-funktiolle parametriksi vastaanottajan jonon. Toiseksi parametriksi annetaan tapahtuma, jonka tietosisältöön on kirjotettu viesti (kuva 2).

Tapahtuman lähetys aloitetaan suorittamalla em_send()-funktio. Nyt tapahtuman lähettämisestä vastaava ohjelmisto tarkastaa, että mihin jonoon lähetetty tapahtuma on menossa.

Uuden tiedonsiirtomekanismin lähetysfunktio

Jos lähetettävä jonotunniste vastaa EM:n ulkopuolista jonoa, tarkastetaan vastaako jonotunniste jonkun viereisen laitteen laitetunnitetta. Jos vastaa, kutsutaan send_turbo_channel_event()-funktiota (kuva 12). Jos ei vastaa, tiedetään, että lähetys on kortin ulkopuolinen, jolloin lähetys tapahtuu BIP:n avulla. (Kuva 11.)

```
em_status_t send_turbo_channel_event(struct rte_mbuf* mbuf, uint32_t remote_queue, EEmSendPriority priority)
{
    uint16_t device_id = (uint16_t)(remote_queue >> 16);
    void* remote_shm_ptr;
    remote_shm_ptr = lookup_turbo_channel_neighbour_shared_mem(device_id);
    if(remote_shm_ptr == NULL) {
        if (!add_turbo_channel_neighbour_device_id(device_id))
        {
            return EM_ERROR;
        }
        remote_shm_ptr = lookup_turbo_channel_neighbour_shared_mem(device_id);
    }
    struct payload_element_t *buf;
    em_event_t event;
    event = em_mbuf_to_event(mbuf);

    bool messageReceived = false;
    while (!messageReceived)
    {
        messageReceived = turbo_channel_shm_alloc(remote_shm_ptr, (void**)&buf);
        if (messageReceived)
        {
            memcpy(buf->user_payload, em_event_pointer(event), em_event_get_size(event));
            buf->em_queue_id = remote_queue;
            buf->size = em_event_get_size(event);
            turbo_channel_queue_push(remote_shm_ptr, buf);
        }
    }
    return EM_OK;
}
```

KUVA 12. Uuden tiedonsiirtomekanismin eventtien lähetyksistä vastaava funktio

Vastaanottojonolle tehdään bittioperaatio, jonka avulla saadaan vastaanottajan EM-instanssin laitetunniste (kuva 12). Tämän avulla voidaan hakea vastaanottajan jaetun muistin alkuosoite, joka on tallennettu aiemmin tietuetaulukkoon (kuva 10).

Seuraavaksi suoritetaan turbo_channel_shm_alloc()-funktio, joka palauttaa osoittimen muistinhallintajonoon. Tapahtuman tietosisältö kopioidaan tietueeseen, joka lähetetään viestijonoon.

Tapahtuman vastaanotto

Kaikki EM:n käsittelijät suorittavat em_poll_rx()-funktiota. Funktion sisällä suoritettava turbo_channel_queue_pop-funktio tarkastaa onko viestejä saapunut jaetun muistin viestijonoon. Jos viesti löytyy, sen tietosisältö kopioidaan ja lähetetään paikallisesti eteenpäin suorittamalla em_send()-funktio. (Kuva 13.)

```
/* Polling events sent between EM instances */
DLL_PUBLIC void em_poll_rx()
{
    struct payload_element_t* buf;
    void* shm_ptr = em_local.turbo_channel_shm_ptr;
    bool messageReceived = false;

    messageReceived = turbo_channel_queue_pop(shm_ptr, (void**)&buf);
    if (messageReceived && buf->size > 0)
    {
        em_event_t event;
        em_queue_t queue = buf->em_queue_id;
        void* payload_ptr;
        event = em_alloc(buf->size, EM_EVENT_TYPE_SW, EM_POOL_DEFAULT);
        payload_ptr = em_event_pointer(event);
        memcpy(payload_ptr, buf->user_payload, buf->size);
        turbo_channel_shm_free(shm_ptr, buf);
        em_send(event, queue);
        messageReceived = false;
    }
}
```

KUVA 13. Viestien pollausfunktio

Tapahtuma saapuu käsiteltäväksi EM:n paikalliseen jonoon. Käsittelijä kutsuu suoritusobjektin eo_receive()-funktiota. Vastaanottaja on saanut tapahtuman ja voi käsitellä sitä haluamallaan tavalla. (Kuva 12.)

4 INTEGROINTI JA TESTAUS

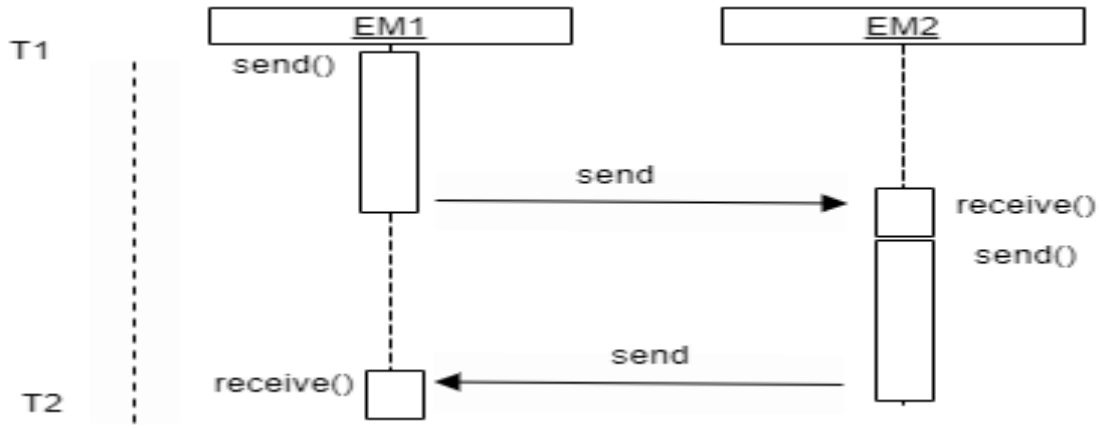
Kun säiliöiden välinen viestintä todettiin toimivaksi EM:n prototyypin päällä, alettiin suunnittelemaan tiedonsiirtomekanismin testaamista.

Testaamista varten tarvittiin kaksi eri säiliöissä olevaa EM-instanssia, joista toinen lähettää viestin ja toinen palauttaa saman viestin takaisin. Tämän perusteella voidaan laskea tapahtuman lähettämisessä kulunut aika ja siirtyvän datan määrä aikayksikköä kohden.

Viestiä lähettävän EM:n tarvitsee ainoastaan tietää vastaanottopuolen jonotunniste, jonka avulla viesti kulkeutuu oikealle vastaanottajalle.

Lähettämisessä kulunut aika eli latenssi saadaan ottamalla aikaleima ennen viestin lähettämistä ja toisen kerran viestin saapuessa takaisin samaan jonoon. Aikaleimojen erotuksesta saadaan tiedonsiirtomekanismin latenssi. (Kuva 14.)

Datansiirtonopeus saadaan jakamalla lähetetyn tiedon määrä lähetykseen kuluneella ajalla. Datansiirtonopeus ilmoitetaan yleensä bitteinä sekuntia kohden tai tavuina sekuntia kohden.



KUVA 14. Tapahtuman kokonaiskierron mittaamiseen kulunut aika T2 – T1

4.1 Alustavat testit

EM:n tiedonsiirrosta oli paljon referenssidataa vanhoista BIP-testeistä. Alustavat latenssitestit tehtiin käyttämällä EM:n prototyyppiä. RTT-testi eli ”round trip time” mitataan siten, että EM lähettää viestin toiselle EM:lle, joka lähettää saman viestin takaisin (kuva 14).

Alustavien testien perusteella vaikutti, että uuden tiedonsiirtomekanismin latenssi olisi huomattavasti huonompi verrattuna BIP:lla tehtyihin RTT-testeihin. Lisämittauksia tehtiin ottamalla aikaleimoja ennen ja jälkeen eri funktioiden suorittamista.

Tapahtumien lähetyspäässä suoritettavat funktiot veivät otettujen mittausten perusteella noin viisinkertaisesti enemmän aikaa, kuin BIP:lla suoritettu RTT-testi.

Tehtiin oletus, että mekanismia suorituskykyä ei voida luotettavasti testata EM-prototyypin ympäristössä. Testausympäristöstö haluttiin samanlaiseksi, missä BIP:lla tehdyt mittaukset suoritettiin.

4.2 Integrointi

Mekanismin koodeja alettiin integroimaan ohjelmiston tuotekehityksessä käytettyyn perushaaraan. Integroinnin seurauksena siirtomekanismin suorituskykytesteistä tuli vertailukelpoisia.

Integrointivaiheessa havaittiin ongelmia, joita ei aiemmin tarvinut ottaa huomioon, kun mekanismia testattiin itsenäisesti toimivan EM:n prototyypissä. Ongelmana oli, että prototyypissä lähetys- ja vastaanottojonojen käynnistäminen suoritettiin manuaalisesti. Tällöin ei tarvinnut ottaa huomioon käynnistyksen synkronointia ja muistialueiden mäppäämistä. Kaikkien EM-instanssien täytyy luoda oma muistinsa ennen kuin voidaan mäpätä viereisten instanssien muistialueet.

Integroinnin tavoitteena oli saavuttaa yhdenvertainen testausympäristö, jossa saavutetut testitulokset ovat vertailukelpoisia. Koodit saatiin integroitua onnistuneesti tuotekehityshaarassa olevaan EM:n koodeihin.

4.3 Verrannolliset mittaukset

Mittaukset tehtiin käyttämällä olemassa olevia BIP-testejä. Muutoksia tehtiin EM:n lähdekoodiin siten, että BIP:n kautta toimivat tapahtumat kulkeutuvat uuden siirtomekanismin kautta.

Alustavat testitulokset olivat lupaavia. Testit osoittivat latenssin olevan pieni verrattuna BIP:lla tehtyihin mittauksiin. Datansiirtonopeus oli kuitenkin huonompi.

Testejä tehtiin kokeilemalla eri parametreja. Testissä ajettavien eventtien kokoa ja lähetyskertoja vaihdettiin, sekä kuinka monta kertaa suorituskykytestit ajettiin.

Datansiirtonopeutta saatiin parannettua vaihtamalla muistinhallintajonon bufferikokoa ja -määrää (kuva 5). Pollauksesta vastaavia funktioita yritettiin optimoida hyödyntämään tiedonsiirtomekanismin ominaisuuksia.

EM:n datansiirtonopeuksien mittaustulokset

Kuvassa 15 nähdään datansiirtonopeusmittaukset BIP:lla (oranssit pisteet) ja uudella tiedonsiirtomekanismilla (purppurat pisteet). Mittaukset tehtiin samalla laitekonfiguraatiolla.

Mittaukset tehtiin kuvan 15 ylemmässä kaaviossa asettamalla lähetettävien eventtien määräksi 1 000 ja kooksi 1 000 tavua. Kuvan 15 alemmassa kaaviossa mittaukset suoritettiin 9 000 tavun eventeillä.

BIP:lla tehdyt 1 000 tavun eventeillä tehdyt mittaukset antoivat kymmenen datapisteen keskimääräiseksi datansiirtonopeudeksi 653 miljoonaa tavua sekunnissa. Uuden mekanismin datansiirtonopeus kolmen mittauksen keskiarvolla oli 529 miljoonaa tavua sekunnissa. Voidaan laskea, että 653 MB/s / 529 MB/s = 1,23. Siis BIP:n datansiirtonopeus on 1,23-kertainen uuteen tiedonsiirtomekanismiin verrattuna.

9 000 tavun eventeillä samoilla ja samoilla datapisteiden määrillä saatiin BIP:n keskimääräiseksi datansiirtonopeudeksi 1,09 GB/s ja uudella tiedonsiirtomekanismilla 895 MB/s. Tiedetään, että 1,09 GB/s = 1 090 MB/s, jolloin saadaan, että

1 090 MB/s / 895 MB/s = 1,22. Siis BIP:n datansiirtonopeus on 1,22-kertainen uuteen tiedonsiirtomekanismiin verrattuna.

KUVA 15. Datansiirtonopeuden mittaustulokset 1 kilotavun ja 9 kilotavun eventeillä

EM:n latenssitestien mittaustulokset

Kuvassa 16 nähdään latenssitestien mittaustulokset BIP:lla (oranssit pisteet) ja uudella tiedonsiirtomekanismilla (purppurat pisteet) suoritetuista mittauksista. Suoritetut mittaukset tehtiin samalla laitekonfiguraatiolla.

Kuvan 16 ylempi kaavio näyttää mittaustulokset 1 000 tavun eventeillä ja alemmassa kaaviossa mittaukset tehtiin 9 000 tavun eventeillä.

Suoritetut mittaukset antoivat keskimääräiseksi latenssiksi 1 000 tavun eventeillä BIP:lla 570 µs ja uudella tiedonsiirtomekanismilla 804 µs. 804 µs / 570 µs = 1,41. Siis uuden tiedonsiirtomekanismin keskimääräinen latenssi on 1,41 kertaa suurempi, kuin BIP:n.

9 000 tavun eventeillä tehdyt keskimääräiset latenssimittaukset antoivat BIP:lla 3,05 ms ja opinnäytetyön mekanismilla 3,71 ms. Saadaan, että 3,71 ms / 3,05

ms = 1,22. Siis opinnäytetyön tiedonsiirtomekanismin latenssi 9 000 tavun eventeillä on 1,22 suurempi kuin BIP:lla tehdyissä mittauksissa.

KUVA 16. Latenssitestien mittaustulokset 1 kilotavun ja 9 kilotavun eventeillä

5. YHTEENVETO

Työn tarkoituksena oli tutkia tukiasemien sisäisessä kommunikoinnissa käytettyä jaettuun muistiin perustuvaa tiedonsiirtomekanismia. Tämä mekanismi olisi vaihtoehto BIP:lle tukiasemien sisäisessä viestinnässä. Työn tarkoitus oli myös perehdyttää työntekijä UP HWAPI -ohjelmiston alijärjestelmiin.

Työssä kehitetty tiedonsiirtomekanismin prototyyppi todettiin lupaavaksi, mutta sitä ei voida tällä hetkellä ottaa käyttöön. BIP:n datansiirtonopeus on noin 1,23-kertainen opinnäytetyössä kehitettyyn tiedonsiirtomekanismiin verrattuna. Latenssitesteissä opinnäytetyön prototyypin oli 1 kB:n eventeillä 1,41-kertainen ja 9 kB:n eventeillä 1,22-kertainen verrattuna BIP:lla tehtyihin mittauksiin.

Integrointi- ja testausvaiheessa todettiin, että suorituskykyä heikentää tapahtuman lähetyspäässä tehtävä tapahtuman kopiointi. Mahdollinen korjausratkaisu olisi suuremmat hugepaget. Tämä mahdollistaisi, että voitaisiin mäpätä EM:n kaikki muistialtaat, jolloin tapahtumien etsiminen EM:n sisäisissä lähetyksissä nopeutuisi.

Jos tulevissa tuotteissa otettaisiin käyttöön suuremmat hugepaget, niin tiedonsiirtomekanismia voitaisiin mahdollisesti käyttää. Työtä voitaisiin jatkokehittää siten, että tapahtuman lähetyksessä dataa ei kopioitaisi ollenkaan.

Opinnäytetyön tulosten perusteella saatiin tietoa jaettuun muistiin perustuvaan prosessien väliseen tiedonsiirtoon liittyvistä mahdollisuuksista ja rajoitteista. Työ antoi myös pohjaa mahdolliselle jatkokehitykselle.

Työ oli haastava ja antoi hyvät perusteet UP HWAPI:n ympäristössä tapahtuvaan ohjelmistokehitystyöhön.

LÄHTEET

1. UP HWAPI Overview. 2021. Sisäinen dokumentti. Nokia Solutions and Networks. Hakupäivä 11.6.2021.

2. Open Event Machine. 2013 . Nokia Siemens Networks. Saatavilla: https://github.com/zlim/open_event_machine. Hakupäivä 28.5.2021.

3. Multicore Event Machine. 2010. Sisäinen dokumentti. Nokia Solutions and Networks. Hakupäivä 11.6.2021.

4. The Interprocess Communication (IPC) Overview. 2018. IBM. Saatavilla: https://www.ibm.com/support/pages/interprocess-communication-ipc-overview. Hakupäivä 28.5.2021

5. Hugepages. 2017. The Debian Project. Saatavilla: https://wiki.debian.org/Hugepages. Hakupäivä 28.5.2021.

6. Understanding memory mapping. 2020. IBM. Saatavilla: https://www.ibm.com/docs/en/aix/7.2?topic=memory-understanding-mapping. Hakupäivä 10.6.2021.

7. What are containers? 2016. Google. Saatavilla: https://cloud.google.com/learn/what-are-containers#section-1. Hakupäivä 8.6.2021.