LAPIN AMK
Lapland University of Applied Sciences

IOT-JÄRJESTELMÄASIANTUNTIJAN PÄIVÄKIRJA

Pulkkinen Juho

Opinnäytetyö

Tieto- ja viestintätekniikka
Insinööri (AMK)

2024

LAPIN AMK
Lapland University of Applied Sciences

Opinnäytetyön tiivistelmä

Tieto- ja viestintätekniikka
Insinööri (AMK)

Tekijä	Juho Pulkkinen	**Vuosi**	2024
Ohjaaja	Tommi Kokko		
Toimeksiantaja	Suvanto Care Oy		
Työn nimi	IoT-järjestelmäasiantuntijan päiväkirja		
Sivumäärä	55		

Työn aiheena on päiväkirjamuotoinen raportointi IoT-järjestelmäasiantuntijan työstäni. Työni koostui tukitehtävistä ja kehitystyöstä, jossa perehdyin IoT-järjestelmän eri osa-alueisiin saavuttaakseni syvällisemmän ymmärryksen järjestelmän hallinnasta ja sen kehityksestä. Opinnäytetyön tavoitteena oli auttaa minua kehittymään IoT-järjestelmäasiantuntijana sekä tutkia, kuinka IoT-järjestelmiä voidaan hyödyntää niin, että ikäihmiset voivat asua pidempään kotona. Lisäksi etsittiin keinoja, joilla IoT-järjestelmään perehdyttämistä voidaan helpottaa.

Tietoperusta kattaa IoT-järjestelmän ja sen keskeiset osa-alueet esitettynä mahdollisimman yksinkertaisesti ymmärrettävässä muodossa. Tutkimusmenetelminä käytettiin eri osa-alueiden dokumentaatioita ja opetusmateriaalia, joiden avulla perehdyttiin järjestelmän eri osa-alueiden toimintaan ja teknisiin yksityiskohtiin.

Työn tuloksena opin ymmärtämään IoT-järjestelmää ja sen osa-alueita paremmin, mikä vahvistaa mahdollisuuksiani hyödyntää opittua työssäni ja syventää osaamistani tulevaisuudessa. Ymmärrän nyt myös paremmin, mitä järjestelmään perehtyminen vaatii, ja voin tämän myötä tukea perehdytysmateriaalin luomista ja kehittämistä. IoT-järjestelmät tarjoavat hoitohenkilökunnalle keinoja seurata ja tukea ikäihmisten arkea sekä ylläpitää yhteyttä heidän kanssaan, mikä voi osaltaan tukea ikäihmisten itsenäistä kotona asumista.

Avainsanat IoT, RTOS, ZigBee

LAPIN AMK
Lapland University of Applied Sciences

Abstract of Thesis

Study Programme in Information
and Communication Technology
Bachelor of Engineering

Author Juho Pulkkinen **Year** 2024
Supervisor Tommi Kokko
Commissioned by Suvanto Care Oy
Title Diary of an IoT System Specialist
Number of pages 55

The aim of this thesis study was to report on the work of an IoT system specialist in diary form. Also, the objective of the study was to develop the author as an IoT system specialist, and to answer the question of how IoT systems can be utilized to enable elderly people to live at home longer.

The work consisted of support tasks and development work. The theoretical foundation covered the components of an IoT system explained in a simple manner. Additionally, ways to create onboarding materials were explored. Research methods included documentation of IoT system components and various educational materials.

The thesis study helped the author to gain a better understanding of IoT systems, which contributed to personal growth as a system specialist. A better understanding of the system assists in the work and in facing future challenges. The study process gave a clearer understanding of what is required to create better onboarding materials. IoT systems are used in healthcare to monitor the daily lives of elderly people and to facilitate communication. Consequently, IoT systems enable elderly individuals to live longer at home.

Key words: IoT, RTOS, ZigBee

SISÄLLYS

1 JOHDANTO

Digitalisaation ja älyteknologian nopea kehitys on avannut uusia mahdollisuuksia sosiaali- ja terveyspalveluiden uudistamiseen, mikä on välttämätöntä kustannusten nousun hillitsemiseksi. Väestön ikääntyessä ennustetaan, että näiden palveluiden kustannukset kasvavat merkittävästi tulevina vuosina. Vaikka palvelujen integraatio ja yhteentoimivuus ovat edistyneet useiden kokeilujen myötä, on ala edelleen kehityksen alkuvaiheissa ja tarjoaa laajan kentän innovaatioille ja parannuksille. (THL 2020, 4.)

Suvanto Care on Rovaniemellä vuonna 2014 perustettu yritys. Suvanto Care vastaa kysyntään, joka on koettu sosiaali- ja terveyspalvelujen toimialalla (Suvanto Care Oy 2024a). Suvanto Carelta löytyy innovatiivisia ratkaisuja kysynnän täyttämiseksi. Yritys on toteuttanut Suvanto Kotona-järjestelmän, joka parantaa ikääntyvien ihmisten mahdollisuuksia asua turvallisesti omassa kodissaan pidempään. Järjestelmän teknologiat eivät ainoastaan lisää asukkaan itsenäisyyttä ja turvallisuutta, vaan mahdollistavat myös kustannustehokkaammat ja kohdennetummat hoitopalvelut. (Suvanto Care Oy 2024b.)

Opinnäytetyön toimeksiantajana toimii Suvanto Care Oy. Työskentelen sillä järjestelmäasiantuntijana. Sain työtehtäväksi tutustua IoT-järjestelmiin ja sen myötä tämä päiväkirjamuotoinen opinnäytetyö keskittyy IoT-järjestelmien tutustumiseen. Päiväkirjamerkinnät kirjoitetaan viikoittain kymmenen viikon ajan, ja niiden kautta dokumentoin oppimisprosessini, johon kuuluvat IoT-järjestelmän eri osa-alueiden ymmärtäminen sekä uusien teknisten tietojen ja taitojen omaksuminen. Opinnäytetyön tavoitteena ja tutkimuskysymyksenä on selvittää, kuinka IoT-järjestelmiä voidaan hyödyntää siten, että ikäihmiset voivat asua kauemmin kotona. Lisäksi tavoitteena on tutustua keinoihin, joiden avulla järjestelmän perehdyttämistä Suvanto Caren henkilökunnalle voidaan helpottaa.

Tämän opinnäytetyön tekstin laatimisessa tekijä on käyttänyt tekoälytyökalua, ChatGPT:n versiota 4o. Tekoälyä on käytetty oikeinkirjoituksen tarkistuksessa sekä tekstin rakenteen muodostamiseen. Tekijä on tarkistanut ja muokannut työkalun laatimaa sisältöä ja ottaa täyden vastuun tekstin sisällöstä.

2 IOT-JÄRJESTELMÄ

2.1 Määritelmä

Internet of Things eli IoT-järjestelmä tarkoittaa laitteiden, sensorien ja ohjelmistojen verkostoa. Internetin välityksellä laitteet voivat jakaa tietoa toisilleen. Laitteet voivat toimia itsenäisesti tai osana laajempaa järjestelmää. Laitteet keräävät, tallentavat, analysoivat ja vaihtaa tietoa keskenään. (Empirica 2024.)

IoT-järjestelmän perusperiaate on, että fyysiset laitteet ja esineet ovat varustettuja sensoreilla, ohjelmistoilla ja muilla teknologioilla, jotka mahdollistavat niiden liittämisen ja kommunikoinnin verkon kautta. IoT-laite voi olla lähes mikä tahansa fyysinen laite, sillä jokseenkin kaikki laitteet ovat muunnettavissa IoT-laitteeksi. IoT-laite on laite, joka voidaan yhdistää Internetiin datan vastaanottamiseksi tai lähettämiseksi. (Empirica 2024.)

2.2 Historia

IoT-järjestelmien alkutaipale voidaan jäljittää 1980-luvulle, jolloin Carnegie Mellon -yliopiston opiskelijat muunsivat juoma-automaatin niin, että automaatin sisältöä voitiin valvoa etänä. Tämä ratkaisu oli kuitenkin kömpelö, ja kehitys eteni hitaasti. Vuonna 2000 LG julkaisi ensimmäisen älyjääkaappinsa, ja vuonna 2008 IoT-laitteiden lukumäärä ylitti jo maailman ihmisväestön lukumäärän. (Empirica 2024.)

COVID-19-pandemian aikana oli kiireellinen tarve kehittää ratkaisuja pandemian hallitsemiseksi. IoT-laitteet tarjosivat tähän tehokkaita vaihtoehtoja, sillä ne pystyvät siirtämään tietoja ilman ihmisen väliintuloa. Tämä teknologia mahdollisti laitteiden yhdistämisen sairaaloihin ja muihin kriittisiin sijainteihin, mikä auttoi pandemiatilanteen hallinnassa ja resurssien seurannassa. (Amon, Kande, Lovett & Ndabeni-Abrahams. 2020, 5.)

IoT-laitteita käytettiin muun muassa tartuntojen varhaiseen tunnistamiseen erilaisilla sovelluksilla, kuten ruumiinlämpöä mittaavilla sensoreilla. Esimerkiksi älylasit oli varustettu lämpökameralla ja kasvojentunnistusteknologialla. Älylasit auttoivat

mittaamaan ihmisten kehon lämpötiloja isommista ryhmistä. Tämä mahdollisti oireellisten henkilöiden tunnistamisen nopeasti ilman suoraa kontaktia. (Castiglione, Umer, Sadiq, Obaidat & Vijayakumar 2021.)

COVID-19-pandemia on korostanut IoT-laitteiden merkitystä jokapäiväisessä elämässä. IoT-laitteiden käyttö pandemian torjunnassa on kuitenkin herättänyt myös huolta niiden turvallisuudesta, yksityisyydestä, yhteensopivuudesta ja tasaarvosta. Näitä kysymyksiä on käsiteltävä laajasti, jotta teknologian hyödyntäminen on sekä turvallista että oikeudenmukaista. (Amon ym. 2020, 5.)

2.3 Tietoturva

IoT-järjestelmien tietoturva on noussut merkittäväksi kysymykseksi laitteiden yleistymisen ja niiden kasvavan roolin vuoksi kriittisissä infrastruktuureissa sekä arkipäivän sovelluksissa. Koska IoT-laitteet ovat jatkuvasti yhteydessä Internettiin ja muihin järjestelmiin, ne ovat alttiita monille tietoturvauhille, kuten tietomurroille, haittaohjelmille ja fyysisille hyökkäyksille. ENISA (European Union Agency for Cybersecurity) tarjoaa suosituksia ja hyviä käytäntöjä IoT-järjestelmien tietoturvan parantamiseksi (Enisa 2019).

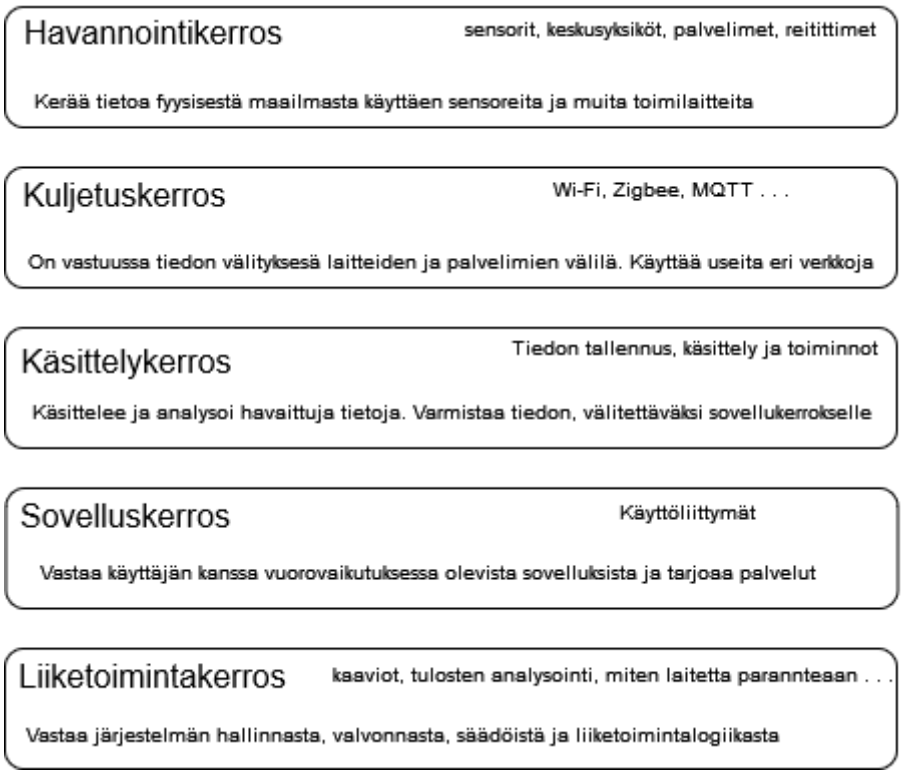
Monissa IoT-laitteissa turvallisuus jää toiminnallisuuden varjoon. Haavoittuvuustestaus on usein puutteellista, ja laitteisiin jää paikkaamattomia haavoittuvuuksia, koska päivitykset ovat monimutkaisia tai niitä ei ole saatavilla ollenkaan. Laitteet jäävät usein toimimaan vanhentuneella laiteohjelmistolla, ja niiden hallinta on heikkoa, erityisesti silloin, kun ne ovat IT-talojen valvonnan ulottumattomissa. (Balbix 2024.)

Yksi merkittävä riski on oletussalasanojen ja heikon tunnistautumisen käyttö, joka altistaa laitteet helposti hyökkäyksille. IoT-laitteiden integroiminen perinteisiin tietoturvajärjestelmiin on haastavaa niiden monimuotoisuuden vuoksi. IoT-laitteiden tuottama valtava tietomäärä voi ylikuormittaa tavanomaiset tietojenkäsittelyjärjestelmät, mikä tekee tiedon eheyden ja turvallisuuden varmistamisesta haastavaa. Koronapandemian myötä lisääntynyt etätyö on laajentanut hyökkäyspinta-alaa,

sillä IoT-laitteet yhdistyvät usein kotiverkkoihin, joissa on heikompi tietoturva. (Balbix 2024.)

2.4 IoT-järjestelmän kerroksittainen arkkitehtuuri ja osa-alueet

IoT-järjestelmän arkkitehtuuri voidaan kuvata kerroksittain (kuvio 1), jolloin järjestelmän osa-alueet jaetaan eri kerroksiin, joilla jokaisella on oma tehtävänsä. Nämä kerrokset toimivat yhdessä muodostaen toimivan kokonaisuuden. Hyvin suunniteltu arkkitehtuuri mahdollistaa saumattoman integroinnin ja hallinnan (ProCoders 2024.) Valitsin viisikerroksisen arkkitehtuurin, jotta se olisi yksinkertaisempi, suoremmin käytännön tilanteisiin sovellettavissa ja helpommin ymmärrettävissä.



Kuvio 1. Arkkitehtuurikuvaus (mukaillen KITRUM 2024)

2.4.1 Havainnointikerros

Havainnointikerrokseen kuuluvat kaikki laitteet ja sensorit, jotka keräävät tietoa ympäristöstä tai suorittavat toimintoja. Järjestelmässä on fyysisiä laitteita, kuten

sensorit tai paikallinen keskusyksikkö. Sensorit havaitsevat ja keräävät tietoa ympäristöstä. (ProCoders 2024.)

Sensorit voivat olla erilaisia mekanismeja tai työkaluja, kuten kamera tai ilmanlaadun mittari. Ne voivat olla integroituina laitteisiin, joiden toimintaa halutaan seurata. Esimerkiksi sensorit voivat mitata vedenpinnan tasoa tai ilman lämpötilaa. (Biba 2023.) Keskusyksiköt toimivat siltana IoT-laitteiden, sensoreiden, Internetin tai pilvipalvelujen välillä (Supermicro 2024).

Esimerkkinä sensorin ja keskusyksikön toiminnasta käytetään ravintolan keittiön jääkaappiin asennettu lämpötilasensori, joka kerää lämpötilatiedot ja välittää ne paikalliselle keskusyksikölle. Keskusyksiköstä tiedot siirretään edelleen Internettiin, jolloin henkilökunnan ei tarvitse pitää lämpötiloista erillistä kirjaa. Henkilökunta voi seurata lämpötiloja sovelluksen kautta.

Toisena esimerkkinä sensorin toiminnasta käytetään potilaiden elintoimintojen valvontaa. Sensorit voivat mitata esimerkiksi sydämen sykettä, verenpainetta tai happisaturaatiota ja lähettää tiedot suoraan keskusyksikön kautta sovellukseen. Lääkärit voivat analysoida tiedot nopeasti sovelluksen avulla, mikä nopeuttaa hoitopäätösten tekemistä ja parantaa hoidon laatua.

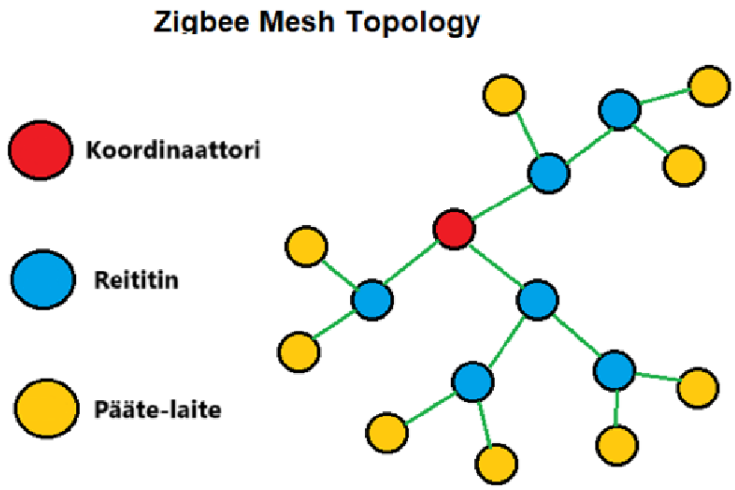
2.4.2 Kuljetuskerros

Kuljetuskerroksella on tärkeä tehtävä tiedon siirtämisessä kerroksesta toiseen. Tässä vaiheessa tieto kulkee erilaisten verkkojen ja protokollien kautta. Kerros yhdistää sensorit, keskusyksiköt ja pilvipalvelut sekä voi suorittaa tiedon reititystä ja pakkausta. (Bellehumeur 2023.)

Kuljetuskerros välittää dataa useista laitteista, kuten sensoreista, kameroista ja toimilaitteista. Data välitetään paikalliseen tai pilvessä sijaitsevaan datakeskukseen (Biba 2024). Kuljetuskerros varmistaa, että tieto liikkuu luotettavasti ja tehokkaasti eri laitteiden välillä (Bellehumeur 2023).

Viestintäteknologioiden protokollia ovat esimerkiksi Zigbee ja LoRaWan. Näitä käytetään erityisesti IoT-sovelluksissa. (Antemijczuk ym. 2019, 16.)

ZigBee on langaton teknologia, joka on kehitetty avoimeksi standardiksi vastaamaan edullisten ja vähävirtaisten langattomien koneiden välisten (M2M) verkkojen tarpeisiin. ZigBee-standardi toimii eri radiotaajuuskaistoilla: 2,4 Gigahertsiä älykotien sovelluksissa maailmanlaajuisesti, 915 Megahertsiä Yhdysvalloissa ja Australiassa, 868 Megahertsiä Euroopassa sekä 784 Megahertsiä Kiinassa. ZigBee-teknologian etuna on mahdollisuus muodostaa mesh-verkkoja (kuvio 2), joissa solmut ovat yhteydessä toisiinsa, ja jokaiselle solmulle on useita reittejä viestien välittämiseksi. Yhteydet päivittyvät dynaamisesti, joten kun yksi solmu sammuu, sen kautta kulkeva reitti ohjataan automaattisesti toisen solmun kautta. Tiedonsiirtonopeus on enintään 250 kilobittiä sekunnissa, ja teoreettinen kantama on jopa sata metriä, mutta käytännössä kantama on usein noin 10 - 30 metriä. (Antemijczuk ym. 2019, 271.)



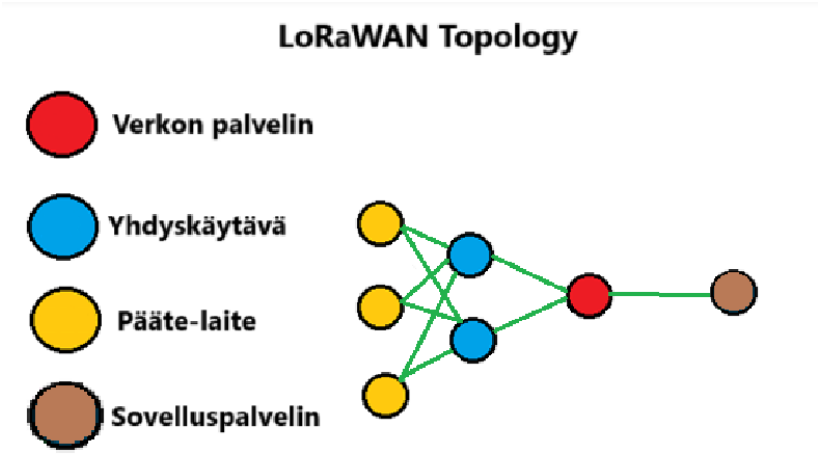
Kuvio 2. Zigbee verkon rakenne (mukaillen Felch 2021)

Päätelaitteet Zigbee-verkossa ovat laitteita, jotka keräävät tietoa ja lähettävät sen koordinaattorille tai reitittimelle. Ne eivät osallistu verkon viestien välittämiseen, vaan ne ovat yksisuuntaisia siinä mielessä, että ne vain lähettävät ja vastaanottavat tietoa. Tämän vuoksi päätelaitteet voivat olla "lepotilassa" säästääkseen virtaa, koska niiden ei tarvitse olla jatkuvasti aktiivisena tiedon reitityksseen. (Walsh 2024.)

Zigbee-verkossa reitittimen pääasiallinen tehtävä on toimia tiedon välittäjinä verkossa. Reitittimet auttavat laajentamaan verkon kantamaa välittämällä viestejä päätelaitteiden ja koordinaattorin välillä. Tämä mahdollistaa sen, että päätelaitteet voivat olla kauempana koordinaattorista. (Walsh 2024.)

Koordinaattori on keskeinen solmu Zigbee-verkossa, ja sen tehtävänä on luoda ja hallita verkkoa. Zigbee-verkossa on vain yksi koordinaattori, joka vastaa laitteiden yksilöllisten osoitteiden jakamisesta, reititystaulukoiden ylläpidosta ja koko verkon toiminnan hallinnasta. (Walsh 2024.)

LoRaWAN on LoRa Alliancen tukema pitkän kantaman tiedonsiirtoteknologia (LoRa). Teknologia tarjoaa suhteellisen alhaisen nopeuden (20 bittiä sekunnissa - 41 kilobittiä sekunnissa) ja kantaman, joka on noin kaksi kilometriä, uusilla lähetinvastaanottimilla dataa voidaan siirtää jopa 15 kilometriä. LoRa käyttää sirpalelevitystekniikkaa 433 Megahertsin ISM-radiotaajuusalueella. Solutopologia on tähtimäinen (kuvio 3), ja yhdyskäytävä sijaitsee keskuspisteessä. Päätesolmut käyttävät yhden hypyn viestintää yhdyskäytävän kanssa, joka on yhteydessä tavanomaiseen IP-verkkoon keskitetyn verkkopalvelimen kautta. LoRaWAN on suunniteltu julkisiin verkkoihin, mutta sitä voidaan käyttää myös yksityisissä verkoissa, jotka eivät vaadi tilausta. (Antemijczuk ym. 2019, 271.)



Kuvio 3. LoRaWAN verkon rakenne (mukaillen Hwan, Kim 2017, 2)

Päätelaiteet LoRaWAN verkossa ovat laitteita, jotka keräävät ja lähettävät tietoa verkkoon. Ne lähettävät tietoa yhden tai useamman yhdyskäytävän kautta verkon palvelimelle, mutta eivät osallistu tiedon välittämiseen verkon muiden solmujen välillä. Päätelaitteet lähettävät viestit kaikille yhdyskäytäville, jotka ovat niiden signaalin kantaman sisällä. (STMicroelectronics 2023.)

Yhdyskäytävä vastaanottaa päätelaitteiden lähettämät viestit ja välittävät ne verkkopalvelimelle. Kaikki yhdyskäytävät, jotka ovat päätelaitteen signaalin kantaman sisällä, vastaanottavat viestin ja lähettävät sen eteenpäin verkkopalvelimelle. Viestejä päätelaitteelle päin lähettäessä verkkopalvelin valitsee parhaan yhdyskäytävän, jonka kautta viesti lähetetään päätelaitteelle. Yhdyskäytävät ovat aina rekisteröitynä tiettyyn verkkopalvelimeen. (STMicroelectronics 2023.)

Verkon palvelin hallitsee viestien välitystä päätelaitteiden ja yhdyskäytävien välillä. Verkkopalvelin vastaanottaa päätelaitteiden viestit yhdyskäytäviltä ja käsittelee ne. Jos sama viesti saapuu useammalta yhdyskäytävältä, palvelin valitsee parhaan version. Verkkopalvelin vastaa myös viestien lähettämisestä päätelaitteille ja valitsee parhaan yhdyskäytävän viestin välittämiseksi. (STMicroelectronics 2023.)

Sovelluspalvelin on pilvessä sijaitseva palvelu, johon päätelaitteiden sovelluskerrokset ovat yhteydessä verkkopalvelimen kautta. Sovelluspalvelin voi olla esimerkiksi ohjauspaneeli, lokipalvelin tai oma protokollaratkaisu. Viestit päätelaitteiden ja sovelluspalvelimen välillä kulkevat eri porttien kautta, jotka on määritelty verkkopalvelimessa. (STMicroelectronics 2023.)

Protokollia sovellusten väliseen tiedonsiirtoon verkon yli ovat esimerkiksi MQTT ja CoAP. Näitä käytetään erityisesti IoT-sovelluksissa. (Antemijczuk ym. 2019, 276-280.)

MQTT (Message Queuing Telemetry Transport) on kevyt viestintäprotokolla, joka on suunniteltu erityisesti laitteiden ja järjestelmien väliseen viestintään, joissa on

rajalliset resurssit, kuten vähäinen kaistanleveys, alhainen virrankulutus ja vähäinen laskentateho. Se on yleisesti käytetty IoT-laitteissa, koska se tukee tehokasta ja luotettavaa viestinvälitystä vähäisillä resursseilla. (MQTT 2024.)
MQTT vaatii toimiakseen keskitetyn välityspalvelimen, joka sijaitsee palomuurien ja NATien ulkopuolella. Kaikki asiakkaat yhdistyvät tähän palvelimeen ja lähettävät sekä vastaanottavat viestejä julkaisija- tai tilaajamallin kautta, jossa viestit lähetetään aiheiden perusteella. (Antemijczuk ym. 2019, 279.) MQTT tekee viestien salaamisen helpoksi käyttämällä TLS:ää ja asiakkaiden tunnistautumisen nykyaikaisilla tunnistautumisprotokollilla (MQTT 2024).

CoAP-protokolla perustuu REST-arkkitehtuuriin ja eroaa MQTT:stä siinä, että se ei käytä keskitettyä palvelinta, vaan jokainen laite toimii omana palvelimenaan tarjoten resursseja asiakkaille. CoAP käyttää UDP-tiedonsiirtoprotokollaa. CoAP on tilaton (*stateless*), eli sen ei tarvitse tallentaa tietoa lähettäjästä tai aiemmista viesteistä, vaan se käsittelee jokaisen viestin itsenäisesti. Tämä vähentää muistin tarvetta ja tekee siitä kevyen ja tehokkaan protokollan. CoAP edellyttää, että jokaisella IoT-laitteella on yksilöllinen tunnus, ja sen avulla voidaan yhdistää laitteita käyttäen standardoituja menetelmiä. CoAP on suunniteltu toimimaan hitaissa IoT-verkoissa, joissa on korkea pakettihävikki, ja se tukee laitteita, jotka voivat olla ajoittain offline-tilassa. (Antemijczuk ym. 2019, 279.)

2.4.3 Käsittelykerros

Käsittelykerros vastaa tiedon tallentamisesta, analysoinnista ja muuntamisesta. Tiedot välitetään käsittelykerrokselle kuljetuskerroksesta. Tämä kerros koostuu pilvipalveluista tai paikallisesti toimivista palveluista, jotka käsittelevät fyysisestä ympäristöstä kerättyä dataa ja tekevät siitä ymmärrettävää. (Bellehumeur 2023.)

Esimerkkejä käsittelykerroksen toiminnoista ovat datan analysointi, reaaliaikainen monitorointi ja ennakoivan huollon mahdollistaminen. Tällä tavalla voidaan esimerkiksi havaita laitteiden ongelmia jo ennen niiden vakavaa vikaantumista.

2.4.4 Sovelluskerros

Sovelluskerros IoT-järjestelmän arkkitehtuurissa sisältää latteiden keräämän tiedon, jotka on koottu yhteen ja niistä muodostettuihin yhteenvetoihin, jotka ovat ihmisten helppoja ymmärtää, kuten kaavioihin ja taulukoihin. Sovelluskerros sisältää myös ohjelmat laitteiden hallintaan ja valvontaan. (Simmons 2022.)

Sovelluskerros kohtaa haasteita tietoturvallisuuden saralla. Erityisesti haasteet liittyvät laitteisiin, joiden laskentateho ja tallennuskapasiteetti ovat heikot. Yleisiä sovelluskerrokseen kohdistuvia tietoturvauhkia ovat esimerkiksi sivustojen välinen komentosarjahyökkäykset ja haitallisen koodin hyökkäykset. (KITRUM 2024.)

2.4.5 Liiketoimintakerros

Liiketoimintakerros hallinnoi koko IoT-järjestelmän toimintaa ja analysoi järjestelmän tuottamaa dataa liiketoiminnan näkökulmasta. Sen tehtävänä on määrittää strategiat ja toimintatavat, jotka perustuvat analysoituun dataan. (Simmons 2022.)

Laitteen menestys riippuu teknologioista ja niiden toimittamisesta kuluttajille. Liiketoimintakerros hoitaa nämä tehtävät laitteen osalta. Tämä sisältää vuokaavioiden ja kaavioiden luomisen, tulosten analysoinnin sekä laitteen parantamismahdollisuuksien arvioinnin ja paljon muuta. (Smartsight 2024.)

3 SEURANTAVIIKOT

3.1 Seurantaviikko 1: Järjestelmäasiantuntijan työ

Aloitan ensimmäisen seurantaviikkoni kertomalla työtehtävistäni, joista työviikkoni muodostui IoT-ympäristössä järjestelmäasiantuntijana. Tavoitteenani on siirtyä entistä enemmän tukitehtävistä osaksi kehitystiimiä. Työni mahdollistaa toimivan ja laajemman IoT-järjestelmäkokonaisuuden. Uusiin IoT-järjestelmän osa-alueisiin tutustun viikoittain näiden töiden ohessa. Työtehtäväni koostuvat monista osa-alueista, jotka yhdessä muodostavat laajemman kokonaisuuden.

3.1.1 Asiakastilaukset

Asiakastilausten toteuttaminen on prosessi, jossa varmistetaan, että asiakkaat saavat tilaamansa laitteet käyttövalmiina ja ajallaan. Tähän prosessiin minun osaltani kuuluu laitteiden konfigurointi, laitteiden liittäminen ylläpitojärjestelmään sekä pakkaaminen ja välittäminen toimitukseen.

Jokainen laite konfiguroidaan asiakkaan vaatimusten mukaisesti. Konfigurointi sisältää käyttöön tarvittavien sovellusten asentamisen, laiteasetusten määrittämisen, ohjelmistopäivitysten asentamisen ja laitteen toiminnan testaamisen. Laitteen tulisi olla käyttövalmis heti asiakkaan vastaanottaessa sen.

Laitteet liitetään ylläpitojärjestelmiin käyttämällä yksilöllisiä tunnistetietoja. Tunnistetietoja voivat olla esimerkiksi sarjanumerot, IMEI-koodit tai jokin muu luotu yksilöivä tunniste, kuten UUID (*universally unique identifier*).

Huolellisesti toteutettu konfigurointi ja laitteiden liittäminen järjestelmään vähentävät asiakastukeen kohdistuvia pyyntöjä ja parantavat käyttökokemusta. Tämä prosessi myös helpottaa laitteiden ylläpitoa, vianmääritystä ja elinkaaren hallintaa.

3.1.2 Asiakaspalvelu

Asiakaspalvelun tukityötehtävät ovat olennaisia IoT-järjestelmän ylläpitotehtäviä, jotka tukevat suoraan asiakkaiden tarpeiden täyttämistä ja ongelmien ratkaisemista. Minun osaltani asiakaspalvelu kohdistuu tekniseen tukeen sekä ongelmanratkaisuun.

Tukipyynnöt voivat liittyä esimerkiksi laitteiden toimintahäiriöihin, kuten laitteiden toimimattomuuteen tai yhteysongelmiin. Kun asiakas ilmoittaa rikkoutuneesta laitteesta, prosessi etenee vaihtoilmoituksen käsittelyyn, jossa vika arvioidaan ja asiakkaalle järjestetään korvaavan laitteen toimitus. Yhteysongelmien korjaaminen etänä on usein haastavampaa. Asiakkaalle voidaan antaa suosituksia mobiiliyhteyden parantamiseksi, kuten laitteen sijainnin muuttaminen tai laitteen vaihtaminen toisen operaattorin liittymällä varustettuun laitteeseen.

3.1.3 Järjestelmän optimointi ja virheenjäljitys

Järjestelmän optimointi on olennainen osa asiakastuen jatkuvaa työtä. Etäyhteyden avulla laitteita voidaan konfiguroida. Konfigurointi parantaa laitteiden suorituskykyä ja mahdollistaa nopean reagoinnin asiakkaiden tarpeisiin.

Asiakastuen tehtäviin kuuluu myös toimintahäiriöiden havaitseminen. Kun ongelmia ilmenee, ne vaativat perusteellista testausta ja virheiden korjausta, jotta järjestelmän luotettavuus ja asiakastyytyväisyys säilyvät korkealla tasolla.

3.1.4 Projektinhallinta, koordinointi ja tiimityö

Työtehtäviini kuuluu viikoittain järjestettäviä etätapaamisia henkilöstön sekä joskus yhteistyötahojen kanssa. Tapaamisissa käsitellään ajankohtaisia asioita, ratkotaan ongelmia, tuodaan esiin haasteita ja tehdään päätöksiä.

Tukitiimin ja tuotannon palaverissa käsittellään avoimien tukipyyntöjen ja tilausten tilannetta. Palaverissa jaetaan työtehtävät ja sovitaan, kuka käsittelee tietyt

tukipyyntöjä ja avoimet tilaukset. Käydään läpi ratkaistut tukipyynnöt ja niiden lopputulokset, ja keskustellaan avoimista ongelmatilanteista sekä havaituista virheistä ja toimintaongelmista. Ennakoidaan myös tulevia haasteita, kuten suurempia päivityksiä ja mahdollisia palvelukatkoksia.

Kehitystiimin palavereissa käsitellään asioita kuten ohjelmistokehityksen ja teknisten projektien edistymistä sekä suunnittelua. Käydään läpi projektien nykytilanteen, toteutuneet saavutukset ja mahdolliset viivästykset. Lisäksi jaetaan uusia tehtäviä ja keskustellaan kuka ottaa vastuun mistäkin tehtävästä.

Koko henkilöstön palaverissa käydään läpi yleiskatsauksen yrityksen tilanteesta, esittellään strategiset tavoitteet ja suunnitelmat sekä uudet käytännöt, työkalut tai teknologiat, jotka vaikuttavat koko henkilöstöön. Lisäksi käsittellään muita ajankohtaisia henkilöstöasioita. Palaverissa on myös tilaa kysymyksille ja avoimelle keskustelulle.

3.2 Seurantaviikko 2: Keskusyksikkö

Seurantaviikolla 2 sovittiin, että työtehtäviini kuuluvat maanantaisin ja perjantaisin tukitehtävät, ja tiistaista torstaihin keskityn kehitystyöhön. Tukitehtäväni keskittyivät asiakkaiden yhteysongelmien selvittämiseen ja niiden ratkaisemiseen. Kun olin saanut selvitettyä odottavat yhteysongelmat, käsittelin vaihtolaitepyyntöjä, joissa rikkinäisten laitteiden tilalle lähetetään korvaava laite.

Kehitystyön tehtävissä keskityin IoT-järjestelmän keskusyksikön toimintaan. Tutustuin keskusyksikön arkkitehtuuriin, sen rooliin tietojen keräämisessä ja hallinnassa sekä siihen, miten se kommunikoi erilaisten sensorien ja laitteiden kanssa. Erityisesti minua kiinnosti, miten datan käsittely tapahtuu tässä vaiheessa.

IoT-järjestelmän keskusyksikkö toimii tiedonkeruupisteenä ja päätöksenteon keskuksena. Se vastaanottaa sensoreiden keräämää tietoa, prosessoi sen, tallentaa ja/tai lähettää sen pilveen jatkoanalyysiä varten sekä reagoi reaaliaikaisesti ta-

pahtumiin. Keskusyksikkö valvoo ja hallinnoi järjestelmää, jotta verkko toimii saumattomasti ja luotettavasti. Protokollat, laitteisto ja tietoturva ovat keskeisiä osia hyvin toimivassa IoT-järjestelmässä.

Sensorit lähettävät tietoa keskusyksikköön, joka vastaanottaa ja käsittelee sen (kuvio 4). Tietoja voidaan käsitellä joko paikallisesti sensorilla, keskusyksikössä tai pilvessä. Mittauksista on hyvä suodattaa tarpeelliset tiedot ja poistaa virheelliset mittaukset. Sensorilta tulevien mittausten suodattaminen ja tarkastelu on tärkeää, jotta data on luotettavaa ja virheetöntä. Suodattamalla dataa voidaan myös vähentää tiedonsiirrossa käytettävää kaistaa ja pitää lähetettävän tiedon määrä mahdollisimman vähäisenä.

```
# Sensorilta saadut tiedot
data = [1, 12, 56, 88, -67, 24.3443, 64.234242]

# Määritellään tiedot, jotka haluamme tallentaa helpommin ymmärrettäväänluettavaan muotoon
sensor_data = {
    "id": data[0],                # Tunniste
    "temperature": data[1],        # Lämpötila
    "humidity": data[2],           # Kosteus
    "battery": data[3],            # Akun varaus
    "signal_strength": data[4],    # Signaalin vahvuus
    "location": {
        "latitude": data[5],       # Sijainti (latitudi)
        "longitude": data[6]       # Sijainti (longitudi)
    }
}
```

Kuvio 4. Sensoreilta tuleva datapaketti ja sen purku

Sensoreilta tuleva tieto voidaan tallentaa tilapäisesti keskusyksikön puskuritietokantaan (kuvio 5), josta tiedot voidaan myöhemmin välittää eteenpäin käytettäväksi. Tällainen toimintamalli mahdollistaa järjestelmän toiminnan jatkumisen, vaikka Internetyhteys katkeaisi. Kun yhteys palautuu, kerätyt tiedot voidaan lähettää eteenpäin ilman tietojen menetystä.

```
# Haetaan tämänhetkinen aikaleima
current_timestamp = datetime.now()

# SQL-lause datan lisäämiseksi tauluun
sql = """
INSERT INTO sensor_data (sensor_id, temperature, humidity, battery,
                         signal_strength, latitude,longitude, timestamp)
VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
"""

# Datan lisäys tauluun, sisältäen aikaleiman
cursor.execute(sql, (data[0], data[1], data[2], data[3],
                     data[4], data[5], data[6], current_timestamp))

# Tallennetaan muutokset tietokantaan
connection.commit()
```

Kuvio 5. Tietojen lisäys tietokantaan.

Mittauksia voidaan käsitellä esimerkiksi asettamalla raja-arvot, tunnistamalla poikkeavat arvot tai käyttämällä loogisia sääntöjä, kuten mittausten muutosnopeuden tarkistamista. Joissakin tapauksissa virheellisten mittausten havaitseminen voi laukaista hälytyksen, mikä on erityisen tärkeää, jos mittausvirheet voivat johtaa merkittäviin ongelmiin, kuten järjestelmän toimintahäiriöihin tai vääränlaisiin automaattisiin toimenpiteisiin.

Mittausten tarkastelun ja suodatuksen ajankohta riippuu käyttötarkoituksesta. Esimerkiksi kuviossa 6 keskusyksikkö voi tehdä reaaliaikaista päätöksentekoa, jos lämpötila-anturi havaitsee, että huoneen lämpötila laskee alle annetun raja-arvon keskellä yötä, keskusyksikkö voi välittömästi käynnistää lämmitysjärjestelmän ja samalla lähettää ilmoituksen käyttäjälle mobiilisovelluksen kautta. Mikäli lämpötilassa ei tapahdu muutoksia, lämpötilatieto voidaan suodattaa pois siinä vaiheessa, kun tiedot välitetään Internetiin. Tämä vähentää sekä tiedonsiirron että järjestelmän kuormitusta.

```
# Tarkistetaan, onko lämpötila alle raja-arvon
    if sensor_data["temperature"] < 16:
        print("Lämpötila alle 16°C! Suoritetaan toimenpiteitä...")
        Käynnistetään_lämmitin()
        lähetä_push_ilmoitus(f"Lämpötila on laskenut alle 16°C, nykyinen lämpötila: {sensor_data["temperature"]} °C.")
```

Kuvio 6. Päätöksentekoa

3.3 Seurantaviikko 3: Reaaliaikaisen käyttöjärjestelmän perusteet

Seurantaviikolla 3 työtehtävieni järjestystä muuttui jälleen. Toimin joka päivä ensimmäiset neljä tuntia tukitehtävissä ja loppupäivän kehitystyössä. Apuani kaivattiin purkamaan aamuista ruuhkaa ongelmien ratkaisemisessa. Tukitehtävät keskittyivät yhteysongelmien ratkaisuun ja vaihtolaitetilausten toimittamiseen. Kehitystyössä keskityin sensorien koodin logiikan ymmärtämiseen ja suunniteltiin, että logiikka tulee dokumentoida. Tehtäväni on hahmotella aktiviteettikaavioita järjestelmän toiminnasta. Näiltä osin en tarkemmin voi asiasta kertoa, joten kerron reaaliaikaisesta käyttöjärjestelmästä.

Reaaliaikainen käyttöjärjestelmä RTOS (*real time operating system*) on aihe, johon tutustuin. RTOS on minulle uusi ja tuntematon. Sen tapa suorittaa koodia eroaa koulusta tutuksi tulleesta superloop-arkkitehtuurista, jossa ohjelman kaikki tehtävät suoritetaan peräkkäin jatkuvassa silmukassa. Sen takia on aiheellista tutustua RTOS:iin.

3.3.1 Johdanto käyttöjärjestelmiin

Järjestelmä voidaan määritellä prosessiksi tai prosessien kokoelmaksi. Käyttöjärjestelmä on ohjelmistojen joukko, joka on suunniteltu hallitsemaan tietokoneen resursseja ja laitteistoa. Esimerkkejä näistä resursseista ovat muisti, syöttö- ja tulostuslaitteet (I/O-laitteet) sekä kommunikaatiolaitteet. Tiedostojärjestelmä, virtuaalimuisti ja tietoturvaominaisuudet ovat käyttöjärjestelmän hallitsemia ohjelmistoresursseja. Käyttöjärjestelmä on ohjelmisto, joka vastaa näiden resurssien hallinnasta. (Embetronicx 2023.)

Ilman käyttöjärjestelmää käyttäjä ei voi suorittaa sovelluksia, ellei ohjelma ole erikseen suunniteltu toimimaan suoraan ilman käyttöjärjestelmää. Käyttöjärjestelmä helpottaa sovellusten suorittamista ja erilaisten ongelmien ratkaisemista, koska sen avulla tietokoneen laitteistoa voidaan käyttää tehokkaasti. (Embetronicx 2023.)

Käyttöjärjestelmän tärkeimpiä tehtäviä ovat prosessinhallinta, muistinhallinta, resurssien hallinta ja syöttö- ja tulostustoimintojen hallinta. Näihin kuuluvat myös sisäisten prosessien hallinta sekä tiedon vastaanoton ja lähettämisen hallinta. (Embetronicx 2023.)

Käyttöjärjestelmä voidaan jakaa kahteen osaan:

- Shell, joka vastaa käyttäjän ja sovellusten hallinnasta ja kommunikoinnista käyttöjärjestelmän kanssa (Embetronicx 2023).
- Kernel, joka tarjoaa perustoiminnot laitteiston hallintaan ja toimii käyttöjärjestelmän ytimenä (Embetronicx 2023).

3.3.2 Reaaliaikaisen käyttöjärjestelmän peruskäsitteet

Reaaliaikainen käyttöjärjestelmä on suunniteltu toimimaan tilanteissa, joissa nopea reagointi on kriittistä virheiden, vääristymien tai katastrofien estämiseksi. Esimerkkejä tällaisista sovelluksista ovat lentoyhtiöiden varausjärjestelmät, työstökoneiden ohjaus ja ydinvoimaloiden valvonta. (Wind River Systems 2024.)

Reaaliaikainen käyttöjärjestelmä (RTOS) on käyttöjärjestelmä, jolla on kaksi keskeistä ominaisuutta: ennustettavuus ja determinismi. RTOS-järjestelmässä toistuvat tehtävät suoritetaan tarkasti määrätyssä ajassa. Tiedetään kuinka kauan tehtävä suorittaminen kestää, koska se tuottaa aina saman tuloksen. Reaaliaikaisen käyttöjärjestelmän etujen vuoksi sitä käytetään usein sulautetuissa järjestelmissä. (Wind River Systems 2024.)

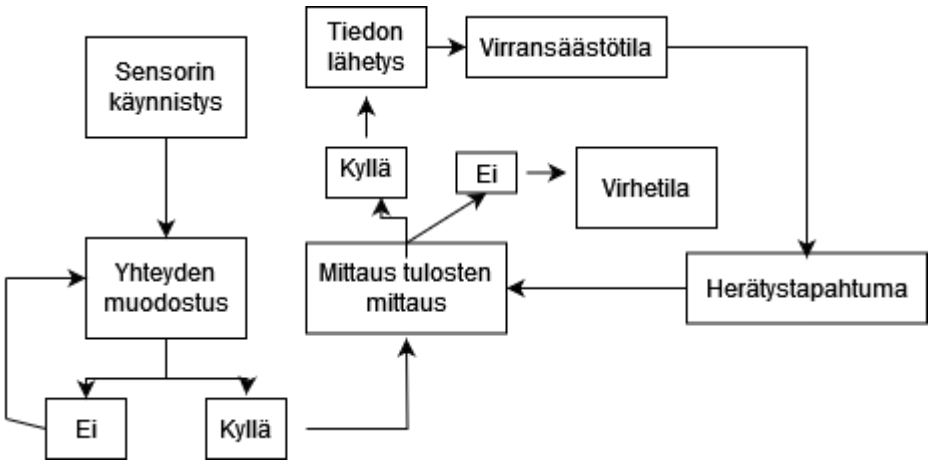
RTOS-järjestelmät jaetaan pehmeisiin (*soft*) ja koviin (*hard*) järjestelmiin. Soft-järjestelmät toimivat muutaman sadan millisekunnin tarkkuudella, ihmisen reaktioajan mittakaavassa. Hard-järjestelmissä sen sijaan vastaukset ovat ennustettavissa kymmenien millisekuntien tai lyhyemmän ajan sisällä. (Embetronicx 2023.)

Reaaliaikaisen käyttöjärjestelmän tehtäviin kuuluu useiden samanaikaisten tehtävien hallinta, mikä varmistaa, että jokainen tehtävä suoritetaan ajallaan ja oikeassa järjestyksessä. Lisäksi järjestelmä varmistaa reaaliaikaisen suorituskyvyn,

jolloin se reagoi kriittisiin tapahtumiin ja tehtäviin täsmällisesti. Reaaliaikainen käyttöjärjestelmä myös hallitsee järjestelmän resursseja, kuten prosessorin ja muistin käyttöä, jakamalla ne tehtävien ja tapahtumien kesken. Keskeytysten käsittely on olennainen osa järjestelmän toimintaa, sillä se mahdollistaa laitteiston keskeytysten nopean käsittelyn ja takaa kriittisiin tapahtumiin välittömän reagoinnin. (Embetronicx 2023.)

3.4 Seurantaviikko 4: Käytännön harjoitustehtävät

Seurantaviikolla 4 työtehtäviini ei tullut muutoksia. Tehtävät olivat samankaltaisia kuin edelliselläkin viikolla. Asiakastuen osalta tehtävät liittyivät yhteysongelmiin. Kehitystyössä jatkoin logiikan oppimista ja dokumentointia. Tutustuessani laitteiston logiikkaan hahmotan siitä aktiviteettikaavioita (kuvio 7), jotka helpottavat logiikan ymmärtämistä.



Kuvio 7. Aktiviteettikaavio (mukaillen Lahane, Pawar 2020, 566)

Jatkoin tutustumista RTOS:iin. Tutustuin aiheeseen käytännön harjoitustehtävillä, joita löysin YouTubesta (Hymel, S. 2021). Harjoituksissa käytettiin ESP32-mikrokontrolleria, joka hyödyntää muunneltua versiota FreeRTOS-järjestelmästä. Tämä mahdollistaa reaaliaikaisen käyttöjärjestelmän toiminnallisuuksien harjoittelun.

3.4.1 Alkuun pääseminen RTOS:in käyttämisessä

Ensimmäisessä harjoituksessa harjoitellaan tehtävän (*task*) luomista. Ohjelma sisältää aloitusfunktion, toistofunktion ja tehtäväfunktiot.

Alustusfunktio käynnistyy aina ensimmäisenä ja toimii tehtävien määrittely- ja rekisteröintipisteenä ajastimelle (*scheduler*). Tehtävät luodaan tehtävän luontifunktiolla ja halutuilla parametreilla. Tehtäväajastin vastaa tehtävien hallinnasta ja aikataulutuksesta. Toistofunktio jää tyhjäksi, koska tehtävät suoritetaan ajastimen avulla, eikä perinteistä jatkuvaa suoritussilmukkaa tarvita.

```
// Pinnit
static const int led_pin = LED_BUILTIN;

// Tehtävä / Toiminto / Funktio: Vilkuta lediä
void toggleLED(void *parameter) {
  while(1) {
    // Muutetaan pinni tilaan "HIGH" = 5v/3.3v / "LOW" = 0v
    digitalWrite(led_pin, HIGH);
    /* Keskeytetään tehtävän suoritus määritetyn ajan verran.
     * Määritellään yhden TICK kestoksi millisekuntti. */
    vTaskDelay(500 / portTICK_PERIOD_MS);
    digitalWrite(led_pin, LOW);
    vTaskDelay(500 / portTICK_PERIOD_MS);
  }
}

void setup() {
  // Määritellään ledin pinni ulostuloksi
  pinMode(led_pin, OUTPUT);

  /* Luodaan tehtävä joka suoritetaan ikuisesti ja joka on määritelty suoritettavaksi
   * tietyllä prosessorin ytimellä */
  xTaskCreatePinnedToCore(
    toggleLED,       // Funktion nimi
    "Toggle LED",    // Tehtävän nimi
    1024,            // Määritellään käytettävän muistin määrä tavuina (Stack size)
    NULL,            // Parametri joka välitetään funktiolle (NULL = tyhjä))
    1,               // Tehtävän tärkeys aste
    NULL,            // Tehtävän käsittelijä (NULL = ei ole))
    app_cpu);        // Suorittimen ydin jolle tehtävä sidotaan suoritettavaksi
}
```

Kuva 8. Ledin vilkutus

3.4.2 Tehtävien aikataulutus

Tehtävien aikataulutus harjoituksessa harjoiteltiin tehtävien suorittamista “rinnakkain”. RTOS-ajastin hallitsee tehtävien suoritusta ja kuinka laitteiston suoritusaikaa jaetaan tehtäville, jotta ne voivat toimia “rinnakkain”. Ajastin päättää jokaisen TICK-aikajakson aikana, mille tehtävälle jaetaan suoritusaikaa. Jos useampi tehtävä on suoritettavassa tilassa samanaikaisesti, korkeammalla prioriteetilla määritetty tehtävä suoritetaan ensin.

Tehtävät (*task*) voivat olla myös eri tiloissa, kuten suorituksessa (running), valmiina suoritukseen (*ready*), odottamassa (*blocked*) tai keskeytettynä (*suspended*). Ajastin antaa suoritusaikaa vain niille tehtäville, jotka ovat valmiustilassa, ja tehtävä siirtyy estettyyn tilaan, jos se odottaa esimerkiksi tapahtumaa tai ajastimen herätystä.

Harjoituksessa on kaksi tehtävää (*task*), jotka tulostavat sarjaterminaaliin tekstiä niin sanotusti samanaikaisesti. Toisella tehtävällä on korkeampi prioriteetti, mutta sitä keskeytetään ajoittain, jotta myös alemman prioriteetin tehtävä saa suoritusaikaa. Lopulta alemman prioriteetin tehtävä poistetaan kokonaan ja jäljelle jää vain yksi tehtävä.

Tehtäville luodaan käsittelijät, joiden avulla tehtävään päästään käsiksi (kuvio 9), sen “ulkopuolelta”. Tämän jälkeen rakennetaan tehtävien logiikka.

```
// Jotain tekstiä tulostettavaksi
const char msg[] = "Olipa kerran punahilkka.";

// Tehtävän käsittelijä
static TaskHandle_t task_1 = NULL;
static TaskHandle_t task_2 = NULL;

// Tehtävä: tulostetaan Sarjaterminaaliin pienemmällä prioriteetillä
void startTask1(void *parameter) {
  // Lasketaan montako kirjainta annetussa tekstissä on
  uint msg_len = strlen(msg);

  /* Ikuinen silmukka jossa silmukoidaan annettu
   * teksti ja tulostetaan yksi kirjain kerrallaan. */
  while (1) {
    Serial.println();
    for (int i = 0; i < msg_len; i++) {
      Serial.print(msg[i]); // Tulostetaan yksi kirjain.
    }
    Serial.println();
    vTaskDelay(1000 / portTICK_PERIOD_MS); // 1 sekunnin viive
  }
}

// Tehtävä: tulostetaan Sarjaterminaaliin korkeammalla prioriteetillä
void startTask2(void *parameter) {
  while (1) {
    Serial.print('*'); // Tulostetaan '*' merkki
    vTaskDelay(100 / portTICK_PERIOD_MS); // 0.1 sekunnin viive
  }
}
```

Kuvio 9. Määrittely ja tehtävien rakentaminen

Tässä tapauksessa alustus- ja toistofunktio ovat itsessään yksi tehtävä, ja ne voidaan nähdä eräänlaisena päätehtävänä (*main*), josta ohjelma aloitetaan. Alustusfunktiossa (kuvio 10) käynnistetään sarjaliikenne ja valmistellaan tarvittavat resurssit, kuten ajastimen ja tehtävien aloituskutsut.

```
void setup() {
  /* Aloitetaan Sarjaliikenne. Määritetään liikenne kulkemaan
   * 300bittiä sekunnissa, että kerkeämme nähdä tehtävien toimintaa. */
  Serial.begin(300);

  /* Odotetaan hetki, että kerkeämme avata
   * terminaalin ja emme menetä tulosteita. */
  vTaskDelay(1000 / portTICK_PERIOD_MS); // Sekunnin viive
  Serial.println();
  Serial.println("---FreeRTOS Harjoitus-------");

  // "Setup ja loop" tehtävän prioriteetti
  Serial.print("Setup and loop task running on core ");
  Serial.print(xPortGetCoreID()); // tulostetaan ytimen tunniste
  Serial.print(" with priority ");
  /* Tulostetaan tällä hetkellä suoritettavan tehtävän prioriteetti.
   * Eli "Setup ja loop" tehtävä */
  Serial.println(uxTaskPriorityGet(NULL));

  // Ilmoitetaan ajastimelle tehtävien aloituksesta
  xTaskCreatePinnedToCore(startTask1,
                          "Task 1",
                          1024,
                          NULL,
                          1,    // Prioriteetti 1
                          &task_1,
                          app_cpu);

  xTaskCreatePinnedToCore(startTask2,
                          "Task 2",
                          1024,
                          NULL,
                          2,    // Prioriteetti 2
                          &task_2,
                          app_cpu);
}
```

Kuvio 10. Ohjelman aloitus

Toistofunktiossa kuviossa 11 käsiteltiin tehtäviä tehtäväkäsittelijöiden kautta. Tehtävä 2 pysäytetään hetkellisesti kahden sekunnin ajaksi, jonka jälkeen suoritusta jatketaan toiset 2 sekunttia.

```
void loop() {
  /* Keskeytetään korkeamman prioriteetin
   * tehtävää tietynajoin hetkellisesti.
   * Silmukka käydään läpi kolme kertaa */
  for (int i = 0; i < 3; i++) {
    vTaskSuspend(task_2);
    vTaskDelay(2000 / portTICK_PERIOD_MS);
    vTaskResume(task_2);
    vTaskDelay(2000 / portTICK_PERIOD_MS);
  }

  /* Kunnes ylempi silmukointi on suopritettu
   * poistetaan alemman prioriteetin tehtävä. */
  if (task_1 != NULL) {
    vTaskDelete(task_1);
    task_1 = NULL;
  }
```

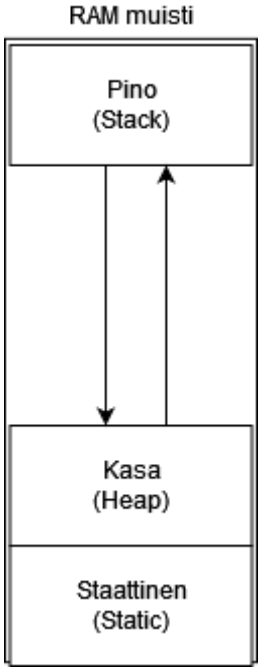
Kuvio 11. Tehtävien keskeytys ja poisto

3.5 Seurantaviikko 5: Käytännön harjoitukset

Seurantaviikolla 5 työtehtäviini kuuluivat asiakastuen tehtävät. Käsittelemäni tehtävät liittyivät suurimmaksi osaksi yhteysongelmiin. Viikon aikana konfiguroin laitteet asiakas- ja vaihtolaitetilauksiin sekä pakkasin laitteet lähetysvalmiiksi. Seurantaviikolla 5 saapui myös uusia ohjelmisto- ja sovellusversioita laitteisiin. Ennen kuin nämä versiot voidaan julkaista tuotantoon, laitteet on testattava huolellisesti, jotta varmistetaan niiden toimivuus ja luotettavuus. Kehitystehtävien osalta viikkoni oli hieman rikkonainen, mutta tehtävät, joihin ehdin paneutua, liittyivät dokumentointiin ja logiikan ymmärtämiseen.

3.5.1 Muistinhallinta

Jatkoin RTOS:iin tutustumista käytännön harjoituksilla. Seurantaviikon 5 ensimmäinen harjoitus käsitteli muistin hallintaa. Ohjelmoinnissa tietokoneen muisti jaetaan osioihin: staattinen, kasa ja pino (kuvio 12). Muistialueiden ymmärtäminen on tärkeää tehokkaan ja virheettömän koodin kirjoittamiseksi. Jos muistialueita ei hallita huolellisesti, ne voivat ylivuotaa, ja pinomuisti- ja kasamuistialueet voivat törmätä, mikä aiheuttaa ongelmia.



Kuvio 12. Muistialueet

Staattinen (*static*) muisti varataan käännösaikana ja säilyy koko ohjelman suorituksen ajan. Se sisältää globaalit muuttujat, vakiot ja funktioiden sisäiset staattiset muuttujat. Staattisten muuttujien muisti varataan, kun ohjelma käynnistyy, ja vapautetaan, kun ohjelma päättyy. (Medium 2024.)

Staattinen muisti tarjoaa vakautta ja pysyvyyttä, mikä tekee siitä sopivan muuttujille, joiden täytyy säilyttää arvonsa useiden funktiokutsujen välillä. Kuitenkin sen kiinteä varaus käännösaikana voi johtaa tehottomuuteen, jos muistia ei hallita oikein, sillä muisti pysyy varattuna, vaikka muuttujaa ei käytettäisi. (Medium 2024.)

Kasa (*heap*) on dynaaminen muistialue, jossa kehittäjät voivat varata muistia suorituksen aikana. Toisin kuin pino, kasa ei noudata tiukkaa rakennetta, mikä mahdollistaa joustavan muistin varaamisen ja vapauttamisen. Kehittäjät ovat vastuussa kasan hallinnasta, ja epäonnistuminen tässä voi johtaa muistivuotoihin tai fragmentaatioon. (Medium 2024.)

Kasa on ihanteellinen tietojen tallentamiseen, kun tiedon koko on tuntematon tai vaihteleva, kuten dynaamiset taulukot tai ohjelman suorittamisen aikana luodut

oliot. Kasa-muistin virheellinen hallinta voi kuitenkin johtaa muistivuotoihin, jolloin varattua muistia ei vapauteta asianmukaisesti, mikä voi vähitellen kuluttaa saatavilla olevan muistin loppuun. (Medium 2024.)

Pino (*stack*) on muistialue, joka toimii periaatteella viimeksi sisään, ensimmäisenä ulos (LIFO). Sitä käytetään tallentamaan paikallisia muuttujia ja funktiokutsuihin liittyvää tietoa. Kun funktiota kutsutaan, sen paikalliset muuttujat ja kutsutiedot työnnetään pinoon. Vastaavasti, kun funktion suoritus päättyy, pino kevennetään, jolloin siihen liittyvä muisti vapautuu. (Medium 2024.)

Pino on nopeaa ja tehokasta, koska muistin varaaminen ja vapauttaminen ovat yksinkertaisia toimenpiteitä. Kuitenkin siinä on rajoituksia. Pinon koko on yleensä rajallinen, ja liiallinen käyttö voi johtaa pinon ylivuotoon (*stack overflow*). Tämän vuoksi pinomuisti soveltuu parhaiten lyhytikäisten muuttujien ja funktiokutsujen hallintaan. (Medium 2024.)

Harjoitus aloitettiin täyttämällä pino-muistialue ja seuraamalla, mitä tapahtuu, kun pino täyttyy liikaa (kuvio 13). Tulosteessa seurataan pinon tilaa funktiolla, joka palauttaa pinon 'korkean vesimerkin' (*high water mark*). Funktion palauttama arvo tarkoittaa pienintä määrää vapaata pinomuistia tehtävän aikana.

```
 * toimenpide muistialueen täyttämiseksi */
void testTask(void *pvParameters) {
  while(1) {
    int a = 1;
    int b[200];

    /* Täytetään Pino (Stack) musitia siten että asetetaan
     * b taulukon jokaiseen alkioon luku 2 */
    for (int i = 0; i < 100; i++) {
      b[i] = a + 1;
    }

    Serial.println(b[0]); // Tulostetaan alkioihin asetettu numero

    // Tulostetaan jäljellä oleva määrä pino muistista
    Serial.print("High water mark: ");
    Serial.println(uxTaskGetStackHighWaterMark(NULL));
```

Kuvio 13. Pino-muistialueen täyttämistä

Odotetusti muistialue täyttyi, ja ohjelman suoritus pysähtyi virheilmoitukseen (kuvio 14). High water mark -funktio ilmoitti, ettei vapaata muistia ole enää käytettävissä.

```
14:15:55.726 -> High water mark: 0
14:15:55.726 ->
14:15:55.757 -> ***ERROR*** A stack overflow in task Test Task has been detected.
14:15:55.790 -> Core  0 register dump:
14:15:55.790 -> MEPC    : 0x40801212  RA      : 0x40805866  SP      : 0x4080ce60  GP      : 0x4080b0e0
14:15:55.790 -> TP      : 0x40805188  T0      : 0x40030dca  T1      : 0x0000000f  T2      : 0x00000001
```

Kuvio 14. Pinon ylivuoto

Lisäämällä pino-muistialueen kokoa tehtävän luonnin yhteydessä saadan ohjelma toimimaan ilman virheilmoitusta. Pino-muistialueen lisäys kuviossa 15.

```
// Luodaan tehtävä annetuin määrittelyin
xTaskCreatePinnedToCore(
    testTask,    // Osoitin toiminnolle joka suoritetaan
    "Test Task", // Tehtävän nimi
    1500,        // Pino (stack) muistialueen koko
    NULL,
    1,
    NULL,
    app_cpu
);
```

Kuvio 15. Pino-muistialueen lisäys

Pino-muistialueen lisäyksen jälkeen tehtävän suoritus onnistuu. Tulosteesta kuviossa 16 näkyy, että vapaata muistia pinossa on taas käytettävissä.

```
14:25:20.224 -> High water mark: 292
14:25:20.305 -> 2
14:25:20.305 -> High water mark: 292
14:25:20.440 -> 2
14:25:20.440 -> High water mark: 292
14:25:20.534 -> 2
```

Kuvio 16. Vapaan muistin määrä pinossa

Tämän jälkeen harjoiteltiin muistialueen varaamista kasasta, jota täytettiin ja seurattiin sen toimintaa. Harjoituksen aikana opittiin myös, kuinka heapmuisti vapautetaan, jotta ohjelma toimisi oikein ja vältyttäisiin muistivuodoilta.

Kasa-muistialueen varaaminen onnistuu c-kielessä malloc-funktiolla (kuvio 17). Ohjelma tulostaa myös vapaan kasamuistin arvon ennen muistin varaamista.

```
/* Tulostetaan vapaan muistin määrä Kasa (Heap) - muistialueelta,
 * ennekuin käytämme Malloc funktiota varaamaan sitä meille. */
Serial.print("Heap before malloc (bytes): ");
Serial.println(xPortGetFreeHeapSize());

/* Varaataan kasasta (Heap) muistia 1024 kokonaisluvun verran, eli 4096 tavua.
 * pvPortMalloc on FreeRTOSin tarjoama funktio muistin varaamiseen. */
int *ptr = (int*)pvPortMalloc(1024 * sizeof(int));
```

Kuvio 17. Kasa-muistialueen varaus

Halutaan havainnollistaa, mitä tapahtuu, kun kasa-muistialueen tila loppuu. Kuviossa 18 esitetään yksinkertainen toiminto, jossa varatun muistialueen alkiot täytetään, ja ohjelma käyttää muistin loppuun. Jokaisen varauksen ja täytön jälkeen ohjelma tulostaa kasa-muistialueen jäljellä olevan vapaan muistin määrän.

```
 * Täytetään varattu muistialueen alkiot luvulla 3 */
for (int i = 0; i < 1024; i++) {
  ptr[i] = 3;
}

/* Tulostetaan vapaan muistin määrä kasa (heap)
 * -muistialueelta Malloc funktion ja alueen täytön jälkeen. */
Serial.print("Heap after malloc (bytes): ");
Serial.println(xPortGetFreeHeapSize());
```

Kuvio 18. Kasa-muistialueen täyttö

Ohjelman suoritus loppui virheilmoitukseen kuviossa 19. Virheilmoitus tarkoitti, että ohjelma yritti päästä käsiksi virheelliseen muistiosoitteeseen.

```
15:15:37.413 -> Heap before malloc (bytes): 17276
15:15:37.413 -> Heap after malloc (bytes): 13164
15:15:37.505 -> 2
15:15:37.505 -> Heap before malloc (bytes): 13164
15:15:37.505 -> Heap after malloc (bytes): 9052
15:15:37.598 -> 2
15:15:37.598 -> Heap before malloc (bytes): 9052
15:15:37.598 -> Guru Meditation Error: Core  0 panic'ed (Store access fault). Exception was unhandled.
15:15:37.632 ->
```

Kuvio 19. Muistin ylivuoto

Malloc-funktio palauttaa arvon NULL, jos muistialueella ei ole riittävästi tilaa. Ohjelman kaatuminen voidaan estää tarkistamalla malloc-funktion palauttama arvo ennen muistialueelle kirjoittamista. Lisäksi ohjelman oikeanlainen toiminta voidaan varmistaa vapauttamalla varattu muistialue käytön jälkeen kutsumalla sitä varten tarkoitettua vapautusfunktiota. (kuvio 20).

```
/* Yksi tapa estää kasan (heapin) ylivuoto on tarkistaa mallocin palautusarvo.
 * Jos muistin varaaminen epäonnistuu, malloc palauttaa NULL. */
if (ptr == NULL) {
  // Jos muistin varaaminen epäonnistuu, tulostetaan virheilmoitus sarjaporttiin.
  Serial.println("Ei tarpeeksi muistia kasassa muistialueella.");

} else {
  /* Jos muistin varaaminen onnistui, käytetään muistia.
   * Täytetään varattu muistialueen alkiot luvulla 3 */
  for (int i = 0; i < 1024; i++) {
    ptr[i] = 3;
  }
}

/* Tulostetaan vapaan muistin määrä kasa (heap)
 * -muistialueelta Malloc funktion ja alueen täytön jälkeen. */
Serial.print("Heap after malloc (bytes): ");
Serial.println(xPortGetFreeHeapSize());

// Vapautetaan varattu muistialue
vPortFree(ptr);
```

Kuvio 20. Muistialueen tyhjennys

3.5.2 Jonojärjestelmä

RTOS käyttää jonojärjestelmää (*queue*) tiedonsiirtoon eri tehtävien ja keskeytysten välillä. Jono tarjoaa tavan kommunikoida ja siirtää dataa järjestelmän eri osien välillä, erityisesti monitehtäväisissä sovelluksissa. Jonot ovat erittäin hyödyllisiä, koska ne mahdollistavat tehtävien synkronoinnin ja tiedon siirtämisen ilman, että järjestelmän eri osat joutuvat aktiivisesti odottamaan toisiaan. Useimmissa tapauksissa jonoa käytetään puskurina, joka toimii “ensimmäisenä sisään, ensimmäisenä ulos”-menetelmän mukaisesti: uudet tiedot lisätään jonon perälle. Tiedot voidaan kuitenkin tarvittaessa myös lisätä jonon etuosaan.

Harjoituksessa opeteltiin jonon luomista ja tehtävien välistä viestintää (kuvio 21). Yksi tehtävä vastasi viestien lukemisesta ja niiden tulostamisesta sarjaterminaaliin, kun taas toinen tehtävä lisäsi viestejä jonoon. Aluksi määriteltiin jonon enimmäiskoko ja käsittelijä, jonka avulla jonoa hallitaan. Tämän jälkeen luotiin tehtävä, joka tarkistaa jonoa sekunnin välein ja lukee sekä tulostaa saapuneet viestit.

```
// Määritellään jonon pituudeksi eneintään 5 viestiä
static const uint8_t viestiJononPituus = 5;

// Jono johon viestit tallennetaan
static QueueHandle_t viestiJono;

/* Funktio tulostustehtävälle: Seurataan ilmaantuuko
 * jonoon viestejä ja tulostetaan se. */
void tulsotaViestit(void *parameters) {
  int saapunutViesti;

  // Ikuinen silmukka
  while(1) {
    /* Tarkistetaan onko viestejä jonossa
     * (ei odotella ilmaantuuko uusia viestejä vaan
     * lueataan ja tulostetaan heti jos viesti löytyy) */
    if (xQueueReceive(viestiJono, (void*)&saapunutViesti, 0) == pdTRUE) {
      Serial.println(saapunutViesti); // Tulostetaan terminaaliin
    }

    // Odotetaan hetki ennen kuin suoritetaan uudelleen
    vTaskDelay(1000 / portTICK_PERIOD_MS);
  }
}
```

Kuvio 21. Jonon luku

Kuviossa 22 luodaan tulostustehtävä ja jono. Tässä tapauksessa alustus- ja toistofunktio toimii toisena tehtävänä, jossa ohjelma lähettää uusia viestejä jonoon puolen sekunnin välein. Näin voidaan seurata, mitä tapahtuu, kun jono täyttyy.

```
  // Luodaan jono
  viestiJono = xQueueCreate(viestiJononPituus, sizeof(int));

  if (viestiJono == NULL) {
    Serial.println("Jonon luonti epäonnistui!");
  }

  // Luodaan tulostus tehtävä
  xTaskCreatePinnedToCore(
    tulsotaViestit,
    "Tulosta viestit",
    1024,
    NULL,
    1,
    NULL,
    app_cpu);
}

void loop() {
  static int uusiViesti = 0;
  /* Yritetään lisätä uusi viesti listaan 10 TICK aikayksikö verran,
   * epäonnistuu jos jono on täynnä */
  if (xQueueSendToBack(viestiJono, (void *)&uusiViesti, 10) != pdTRUE) {
    Serial.println("Jono on täynnä");
  }
  uusiViesti++;

  // Odotetaan hetki ennen kuin yritetään uudelleen
  vTaskDelay(500 / portTICK_PERIOD_MS);
```

Kuvio 22. Tehtävän ja jonon luonti sekä jonoon lähetys

Kuviossa 23 huomataan, että jono täyttyy, koska ohjelma lähettää sinne viestejä nopeammin kuin ehtii lukea niitä. Joka kerta, kun viesti luetaan, vapautuu tilaa uudelle viestille.

```
20:15:23.029 -> ---FreeRTOS Jono Harjoitus---
20:15:23.993 -> 0
20:15:24.993 -> 1
20:15:25.993 -> 2
20:15:27.035 -> 3
20:15:27.544 -> Jono on täynnä
20:15:27.993 -> 4
20:15:28.529 -> Jono on täynnä
20:15:29.037 -> 5
20:15:29.539 -> Jono on täynnä
```

Kuvio 23. Jono täyttyi

Jos halutaan optimoida jono niin, ettei se täyty, voidaan esimerkiksi säätää viiveitä viestien lukemisen tai lähettämisen välillä. Jonon kokoa voi myös muuttaa liian pieni jono täyttyy nopeasti, kun taas liian suuri jono kuluttaa turhaan muistia.

3.6 Seurantaviikko 6: Lukitusobjekti ja semafori

Seurantaviikolla 6 kehitystyöhön käytetty aika väheni kahden työntekijän poissaolon vuoksi. Työtehtäväni keskittyivät tukipyyntöjen käsittelyyn, palautuvien laitteiden vastaanottoon ja tarkistukseen sekä laite- ja asiakastilausten tekemiseen. Loppuviikosta sain tehtäväkseni testata ’Suvanto Kotona’ -järjestelmän uudistetun keskusyksikön toimivuutta.

Tutustuin RTOS-järjestelmän lukitusobjekti- ja semafori-synkronointityökaluihin, joita käytetään hallitsemaan säikeiden tai tehtävien pääsyä ja vuorovaikutusta jaettujen resurssien, kuten muistin, oheislaitteiden tai muiden kriittisten alueiden kanssa. Niiden päätarkoitus on estää ”kilpailutilanteet” ja varmistaa oikea ja turvallinen toiminta, kun useat tehtävät voivat yrittää käyttää samoja resursseja samanaikaisesti.

3.6.1 Lukitusobjekti

Lukitusobjekti (*mutex*) toimii kuin avain, joka rajoittaa pääsyn resurssiin. Jos tehtävä haluaa käyttää suojattua resurssia, sen täytyy ensin saada avain (*mutex*). Jos avain on jo jonkun muun käytössä, tehtävän täytyy odottaa, kunnes avain vapautuu. Kun tehtävä saa avaimen, se omistaa sen ja vapauttaa avaimen vasta, kun se ei enää tarvitse suojattua resurssia.

Kuvitellaan yritys, jossa kolme työntekijää jakaa yhden yhteisen auton. Tässä tapauksessa auto on jaettu resurssi, ja auton avain toimii lukitusobjektina. Jos työntekijä haluaa käyttää autoa, hänen täytyy saada auton avain (lukitusobjekti). Jos joku toinen työntekijä on jo ottanut avaimen ja käyttää autoa, muiden on odotettava, kunnes avain palautuu, ennen kuin he voivat käyttää autoa.

Harjoituksessa on kaksi tehtävää, jotka molemmat käyttävät samaa funktiota, joka toimii yksinkertaisena laskurin korottajana (kuvio 24). Ohjelma suoritetaan tarkoituksella ensimmäisellä kerralla virheellisesti, jotta voidaan havainnollistaa, kuinka mutex korjaa tilanteen.

```
// Tämä muuttuja on kriittinen, koska molemmat tehtävät voivat yrittää muuttaa sitä.
int sharedCounter = 0; // Globaali jaettu resurssi - yksinkertainen laskuri

// Funktio, jota molemmat tehtävät käyttävät
void incrementCounter(void *pvParameters) {
  while (1) {
    int localCounter = sharedCounter;
    delay(500);  // Viive ennen päivitystä, simuloi kilpailutilannetta
    localCounter++;                    // Kasvatetaan laskuria
    sharedCounter = localCounter;      // Päivitetään jaettu laskuri
    Serial.print("Counter: ");
    Serial.println(sharedCounter);
  }
}
```

Kuvio 24. Funktion toiminnallisuus

Kuviossa 25 näkyy ohjelman nykyinen suoritus. Toisella tehtävällä on hallussaan vanhentunut arvo jaetusta muuttujasta, vaikka toinen tehtävä on jo korottanut sen.

```
-> ----FreeRTOS kilpailutilanne harjoitus----
-> Task 1 on ajossa
-> Task 2 on ajossa
-> Counter: 1
-> Counter: 1
-> Task 1 on ajossa
-> Task 2 on ajossa
-> Counter: 2
-> Counter: 2
```

Kuvio 25. Ohjelman tulostus

Ohjelman suoritus halutaan toteuttaa siten, että jokaisella tulostuksella laskurin arvo kasvaa yhdellä, eikä tehtävillä ole käytössään vanhentunutta tietoa. Kuviossa 26 ohjelmaan lisätään lukitusobjekti, jotta ohjelma toimisi halutulla tavalla.

```
// Määritellään mutex
SemaphoreHandle_t xMutex;

// Funktio, jota molemmat tehtävät käyttävät
void incrementCounter() {
  // Otetaan mutex käyttöön ennen jaetun resurssin käsittelyä
  if (xSemaphoreTake(xMutex, portMAX_DELAY) == pdTRUE) {
    int localCounter = sharedCounter;
    delay(500);             // Simuloidaan viivettä
    localCounter++;         // Kasvatetaan laskuria
    sharedCounter = localCounter;
    Serial.print("Counter: ");
    Serial.println(sharedCounter);

    // Vapautetaan mutex
    xSemaphoreGive(xMutex);
  }
}
```

Kuvio 26. Mutexin määritys ja käyttöönotto funktiossa

Lukitusobjekti (*mutex*) täytyy myös luoda ohjelman aloituksessa. Alustusfunktioon lisätään lukitusobjekti kahvan kautta suoritettava luontifunktio (kuvio 27).

```
// Luodaan mutex
xMutex = xSemaphoreCreateMutex();

// Luodaan kaksi tehtävää
if (xMutex != NULL) {
  // Käynnistetään ensimmäinen tehtävä, joka tekee lisäyksiä jaettuun muuttujaan.
  xTaskCreatePinnedToCore(
    Task1,   // Funktio, joka suoritetaan.
    "IncTask 1",        // Tehtävän nimi.
    1024,               // Tehtävän pino koko (1024 tavua).
    NULL,               // Ei parametreja tehtävälle.
    1,                  // Tehtävän prioriteetti (1).
    NULL,               // Ei tehtävän kahvaa, joten NULL.
    app_cpu             // Prosessoriydin, johon tehtävä on kiinnitetty.
  );

  // Käynnistetään toinen tehtävä, joka tekee lisäyksiä jaettuun muuttujaan.
  xTaskCreatePinnedToCore(
    Task2,   // Sama Funktio kuin ensimmäisessä.
    "IncTask 2",        // Toinen tehtävän nimi.
    1024,               // Tehtävän pino koko (1024 tavua).
```

Kuvio 27. Lukitusobjektin ja tehtävien luonti ohjelman aloituksessa

Lukitusobjektin lisäämisen jälkeen voidaan todeta kuviosta 28, että ohjelma toimii halutulla tavalla. Laskuri kasvaa jokaisella tulostuskerralla yhdellä, ja tehtävillä on aina ajantasaista tietoa muuttujasta.

```
-> ----FreeRTOS kilpailutilanne harjoitus----
-> Task 1 on ajossa
-> Task 2 on ajossa
-> Counter: 1
-> Task 1 on ajossa
-> Counter: 2
-> Task 2 on ajossa
-> Counter: 3
-> Task 1 on ajossa
-> Counter: 4
```

Kuvio 28. Haluttu tulos

3.6.2 Semafori

Semaforeja käytetään hallitsemaan pääsyä jaetuille resursseille tai synkronoimaan tapahtumia. Semaforeja käytetään usein ympäristöissä, joissa on useita tehtäviä, varmistamaan, että resurssit eivät joudu usean tehtävän samanaikaiseen käsittelyyn, tai että tehtävät suorittavat toimintoja oikeassa järjestyksessä.

Binääristä semaforia käytetään kahteen pääasialliseen tarkoitukseen: varmistamaan, että jaettu resurssi on vain yhden tehtävän käsittelyssä ja synkronointiin. Binääriset semaforit ovat yksinkertaisia "lukitusmekanismeja", joilla hallitaan pääsyä yhteisiin resursseihin tai synkronoidaan tapahtumia tehtävien tai keskeytysten välillä. Binääriset semaforit ja lukitusobjektit (*mutex*) ovat hyvin samanlaisia, mutta niissä on joitain eroja. Lukitusobjektit sisältävät prioriteetin perintämekanismin, kun taas binääriset semaforit eivät. Tämä tekee binäärisistä semaforeista paremman valinnan synkronointiin. (Freertos 2024a.)

Binäärinen semafori toimii kuten kytkin, jolla voi olla vain kaksi tilaa "annettu" tai "otettu". Näin ollen se voi olla joko tyhjä (0) tai täysi (1). Tehtävä tai keskeytys voi "ottaa" semaforin (asettaa sen tilan 0) tai "antaa" sen (asettaa sen tilan 1).

Laskuri-semafori (*counting semaphore*) on synkronointimekanismi, joka mahdollistaa useamman kuin yhden tapahtuman tai resurssin hallinnan. Se eroaa binäärisestä semaforista siten, että laskurisemafori voi pitää kirjaa useista samanaikaisista tapahtumista tai resursseista. (freertos 2024b.)

Laskurilukkoja käytetään tyypillisesti kahteen tarkoitukseen. Ensimmäinen esimerkki käyttötapauksesta, jossa tapahtumakäsittelijä "antaa" lukon jokaisen tapahtuman tapahtuessa (kasvattamalla lukon arvoa), ja käsittelijätehtävä "ottaa" lukon aina käsitellessään tapahtuman (vähentäen lukon arvoa). Lukon arvo edustaa siis tapahtumien ja käsiteltyjen tapahtumien välistä eroa. Tässä tapauksessa on toivottavaa, että lukon arvo on (0), kun lukko luodaan. (freertos 2024b.)

Toisessa esimerkissä lukon arvo ilmaisee saatavilla olevien resurssien määrän. Tehtävän on saatava lukko (vähentämällä lukon arvoa) hallitakseen resurssia. Kun lukon saavuttaa arvon (0), vapaita resursseja ei ole. Kun tehtävä lopettaa resurssin käytön, se "antaa" lukon takaisin (kasvattamalla lukon arvoa). Tässä tapauksessa on toivottavaa, että lukon arvo on maksimimäärä, kun lukko luodaan. (freertos 2024b.)

3.7 Seurantaviikko 7: Ajastimet

Seurantaviikolla 7 käytiin työmatkalla Oulussa tarkistamassa laitteiden toimintaa paikan päällä. Viikon muina päivinä työtehtäväni jakautuivat normaalisti tukitehtävien ja kehitystyön välillä. Kehitystehtäviini kuului tutustua cellular-modeemin toimintaan ja sen käyttäytymiseen 3G-verkon alasajon jälkeen. Lisäksi perehdyin Zigbee-verkon muodostamiseen ja siihen, kuinka Wi-Fi-verkko voi häiritä Zigbee-verkon toimintaa.

RTOS-järjestelmän osalta tutustuin ajastimiin. Ajastimet (*timers*) ovat olennainen osa RTOS-järjestelmiä, koska ne mahdollistavat tehtävien suorittamisen tietyin aikavälein tai tietyn ajan kuluttua. Ajastimia voidaan käyttää tehtävien aikatauluttamiseen, viiveiden luomiseen ja moniin muihin toimintoihin.

Harjoituksessa syötetään sarjaporttiin kirjaimia. Kun ensimmäinen kirjain vastaanotetaan, LED syttyy ja ajastin käynnistyy. Jokainen uusi syöte nollaa ajastimen laskurin, jolloin ajastimen aika alkaa alusta. Jos sarjaporttiin ei tule enää uusia syötteitä, ajastin sammuttaa LEDin viiden sekunnin kuluttua.

Ajastimen ja tehtävän määrittely kuviossa (kuvio 29). Ajastimessa on callback-toiminto, joka suoritetaan, kun ajastin laukeaa, ja se voi esimerkiksi käynnistää tietyn tehtävän, joka tässä tapauksessa on LEDin sammutus.

```
one_shot_timer = xTimerCreate(
  "One-shot timer",           // Ajastimen nimi
  5000 / portTICK_PERIOD_MS,  // Ajastimen aika: 5000 ms
  pdFALSE,                    // Toistuuko ajastin automeettisesti = FALSE
  (void *)0,                  // Ajastimen tunniste (ID)
  turnOffLed                  // Funktio, joka kutsutaan, kun ajastin laukeaa
);

if (one_shot_timer != NULL) {
  xTaskCreate(
    readSerial,
    "readSerial",
    1024,
    NULL,
    1,
    NULL
  );
} else {
  Serial.println("Ajastimen luominen epäonnistui!");
```

Kuvio 29. Ajastimen määrittäminen ja tehtävän luominen

Ajastimen lauettua suoritetaan yksinkertainen callback-funktio, joka vain sammuttaa LEDin (kuvio 30). Callback-funktioon viitattiin ajastimen määrittelyn yhteydessä.

```
void turnOffLed(TimerHandle_t xTimer) {
  digitalWrite(ledPin, LOW);      // Sammutetaan ledi
}
```

Kuvio 30. Callback-funktio

Jokainen luettu kirjain tallennetaan muistiin myöhempää tulostusta varten. Jokaisen kirjaimen kohdalla ajastin joko käynnistyy uudelleen tai nollautuu. Jos syötteestä löytyy välilyönti tai enter, merkkijono null-terminoidaan, mikä osoittaa merkkijonon loppukohdan. Lopuksi indeksi ja puskuri nollataan uutta syötettä varten, jolloin voidaan käsitellä uusi syöte.

```
void readSerial(void *pvParameters) {
  uint8_t bufferLength = 255;     // Muistin koko merkkeinä (tavuina)
  char inputBuffer[bufferLength]; // Alustetaan muisti ja kerrotaan alueen koko
  char incomingByte ;             // Muuttuja tallentamaan kirjaimen
  uint8_t index = 0;   // Indeksillä pidetään kirjaa missä kohtaa muistia ollaan

  while (1) {
    if (Serial.available() > 0) {
      /* Jos sarjasyötteeltä löytyy luettavaa,
       * käynnistetaan tai nollataan ajastin. */
      xTimerStart(one_shot_timer, 0);
      digitalWrite(ledPin, HIGH);          // Sytytetään LED
      incomingByte = Serial.read();        // Luetaan kirjain sarjaportilta
      inputBuffer[index] = incomingByte ;  // Tallennetaan se muistiin
      index++;                             // Kasvatetaan indeksiä

      // Tarkistetaan, onko uusi rivi (enter) tai bufferi täynnä
      if (incomingByte  == '\n' || index >= (bufferLength - 1)) {
        inputBuffer[index] = '\0';      // Null-terminoidaan merkkijono
        Serial.println(inputBuffer);    // Tulostetaan sarjasyöte
        memset(inputBuffer, 0, bufferLength);  // Nollataan bufferi
        index = 0;                      // Nollataan indeksi
      }
    }
    // Viivytetään 500ms, ettei kuormiteta prosessoria liiaksi.
    vTaskDelay(500 / portTICK_PERIOD_MS);
  }
```

Kuvio 31. Ohjelman logiikka

3.8 Seurantaviikko 8: Keskeytykset

Seurantaviikolla 8 osallistuin normaaliin tapaan tuen toimintaan. Helpottaakseni tukitehtävien ruuhkaa tikettijärjestelmässä ratkaisin häiriöilmoituksia laitteiston ja yhteysongelmien osalta. Kehitystyössä perehdyin Zigbee-järjestelmän koordinaattorin toimintaan.

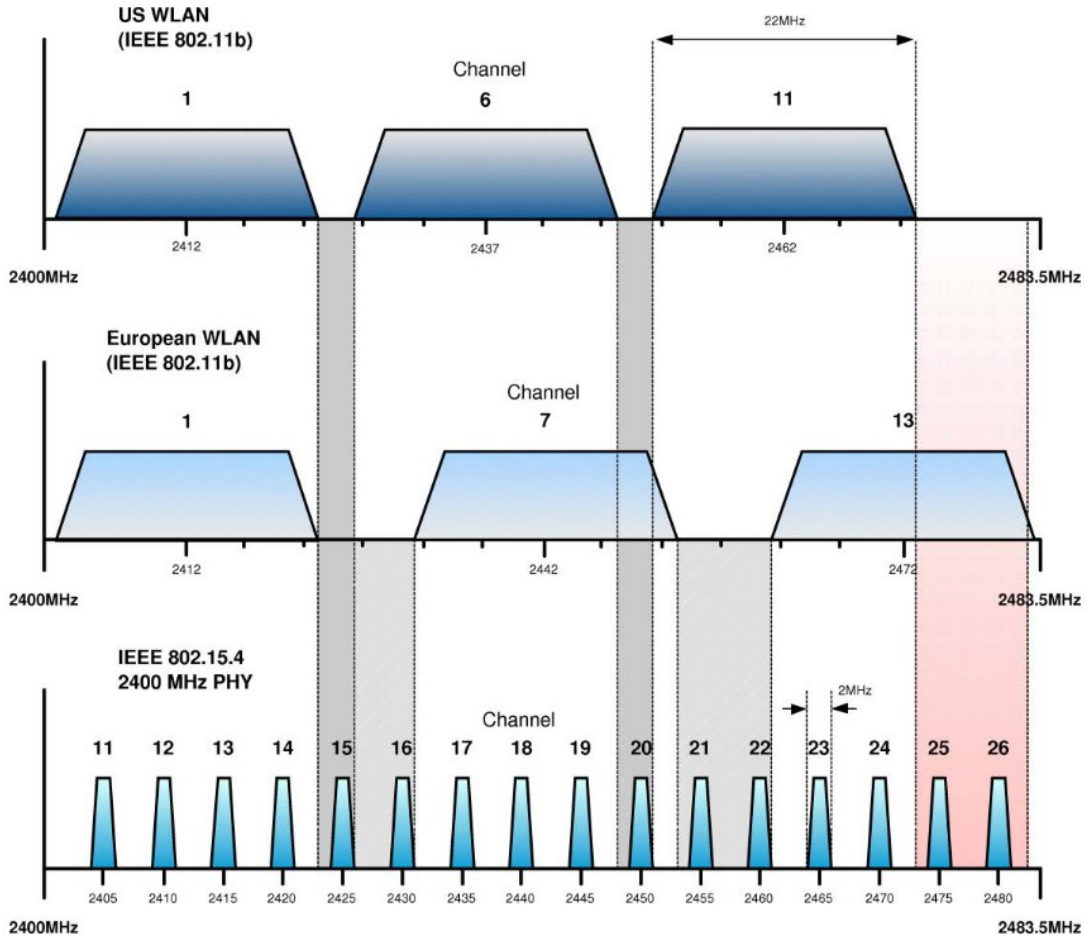
3.8.1 Zigbee-verkon haasteet

Tutustuin Zigbee-verkon muodostamiseen ja sen yhteydessä valittaviin verkon kanaviin. Zigbee-verkko toimii samalla 2,4 Gigahertsin taajuusalueella kuin Wi-Fi- ja Bluetooth-verkot sekä jopa mikroaaltouuni (NXP 2013, 22).

Zigbee-verkko käyttää 2,4 Gigahertsin taajuusaluetta 2405 Megahertsin ja 2480 Megahertsin väliltä. Tältä alueelta Zigbee määrittelee 16 kanavaa, joista käytössä ovat kanavat 11 – 26. Yhden kanavan taajuuskaista on leveydeltään 2 Megahertsiä. (Henderson 2024.)

Wi-Fi-verkko jakaa 2,4 Gigahertsin taajuusalueen 14 kanavaan, mutta kaikki niistä eivät ole käytettävissä kaikissa maissa. Yleisimmin käytettyjä kanavia Euroopassa ovat 1, 7 ja 13, jotka sijoittuvat alueelle 2412 – 2472 Megahertsiä. Yhden Wi-Fi-kanavan taajuuskaistan leveys on 22 Megahertsiä. (NXP 2013, 7, 10, 24.)

Vaikka verkkojen kanavien numerointi saattaa antaa vaikutelman, etteivät ne ole päällekkäin, kuviosta 32 käy ilmi, että numerointi ei estä kanavien päällekkäisyyttä ja verkon häiriöitä. Tätä tilannetta voidaan helpottaa ottamalla ZigBee-verkossa käyttöön kanavat, jotka ovat vähiten päällekkäisiä Wi-Fi-verkon kanavien kanssa. ZigBee-verkon kanavat 15, 16, 21 ja 22 eivät mene päällekkäin Euroopassa käytettyjen Wi-Fi-kanavien kanssa. (NXP 2013, 20.)



Kuvio 32. Kanavien päällekkäisyys (NXP 2013, 11)

3.8.2 RTOS-järjestelmän keskytykset

RTOS-järjestelmästä tutustuin keskeytyksiin ISR (*interrupt service routine*). Keskeytykset ovat olennainen osa reaaliaikaista järjestelmää, ne mahdollistavat nopean reagoinnin ulkoisiin tai sisäisiin tapahtumiin ilman, että prosessori joutuu odottamaan aktiivisesti.

Esimerkkiharjoituksessa käytetään keskeytystä herättämään korkeamman prioriteetin tehtävä, joka sytyttää tai sammuttaa LEDin. Matalamman prioriteetin tehtävä tulostaa sarjaterminaaliin LEDin nykyisen tilan kymmenen kertaa sekunnissa. Keskeytyskäsittelijällä on korkein prioriteetti, mikä estää muiden tehtävien suorituksen ja varmistaa, että keskeytys käsitellään aina välittömästi.

Harjoituksessa esitellään ensin mikrokontrollerissa käytetyt pinnit LEDille ja painonapille (kuvio 33). Globaaleihin muuttujiin määritellään LEDin nykyinen tila sekä binääriselle semaforille "kahva", jonka kautta semaforiin päästään käsiksi. Määritellään keskeytykselle funktio, joka määrittää toiminnot, kun keskeytys havaitaan. Keskeytysfunktio herättää korkeamman prioriteetin tehtävän ja vapauttaa binäärisen semaforin eli niin sanotun lukon.

```
// Pinnit
#define BUTTON_PIN 46
#define LED_PIN LED_BUILTIN

// Globaalit muuttujat
static uint8_t currentLedState = HIGH;
// Binaarinen semafori käytettäväksi ISR:ssä
static SemaphoreHandle_t binarysemaphore = NULL;

// Interrupt Service Routine (ISR)
void IRAM_ATTR onButton() {
    /* Muuttuja, joka kertoo, pitääkö korkeamman
     * prioriteetin tehtävä herättää ISR:n jälkeen */
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;

    /* Annetaan semafori ISR:stä ja tarkistetaan, täytyykö korkeamman
     * prioriteetin tehtävä herättää suoritettavaksi */
    xSemaphoreGiveFromISR(binarysemaphore, &xHigherPriorityTaskWoken);

    // Suorita korkeamman prioriteetin tehtävä välittömästi, jos se on herätetty
    if (xHigherPriorityTaskWoken == pdTRUE) {
        portYIELD_FROM_ISR();
    }
  }
```

Kuvio 33. Keskeytysfunktio

Seuraavaksi määritellään tehtävät (kuvio 34). Ensimmäinen tehtävä odottaa keskeytyksen antamaa binääristä semaforia, jonka saatuaan se joko sytyttää tai sammuttaa LEDin. Tämä tehtävä määritellään korkeammalle prioriteetille kuin toinen tehtävä, jotta se suoritetaan välittömästi keskeytyksen tapahtuessa. Muutoin tehtävä on estettynä, eikä sitä suoriteta, joten se ei kuluta järjestelmän resursseja. Toinen tehtävä tulostaa LED-pinnin nykyisen tilan kymmenen kertaa sekunnissa, jotta toimintaa voidaan seurata ja varmistaa, että keskeytykset tapahtuvat ja korkeamman prioriteetin tehtävä sytyttää tai sammuttaa LEDin

```
void toggleLED(void *pvParameters) {
    while (1) {
        // Odotaa semaforia (ISR:ltä)
        if (xSemaphoreTake(binarysemaphore, portMAX_DELAY) == pdTRUE) {
            currentLedState ^= 1; // Vaihda LEDin tila (0 -> 1 tai 1 -> 0)
            digitalWrite(LED_PIN, currentLedState); // Päivitä uusi tila LEDille
        }
    }
}

void printSerial(void *pvParameters) {
    while (1) {
        // Tulostetaan LEDin tila sarjaporttiin
        Serial.print("LED tilassa: ");
        Serial.println(currentLedState);
        // Odota 100 ms ennen seuraavaa tulostusta
        vTaskDelay(100 / portTICK_PERIOD_MS);
    }
}
```

Kuvio 34. Tehtävät

Setup-funktiossa (kuvio 35) luodaan aiemmin määritellyt semaforit ja tehtävät sekä asetetaan keskeytys kuuntelemaan tapahtumaa, joka syntyy napin painalluksesta. Setup-funktio suoritetaan kerran ohjelman käynnistyessä, minkä jälkeen sitä ei enää tarvita.

```
void setup() {
    // Alustetaan sarjaportti nopeudella 115200 bps
    Serial.begin(115200);
    // Odotetaan 1 sekunti, jotta sarjaliikenne ehtii käynnistyä
    vTaskDelay(1000 / portTICK_PERIOD_MS);
    Serial.print("---RTOS ISR HARJOITUS---");

    // Aseta painonappi syöttötilaan, jossa on ylösvedon vastus
    pinMode(BUTTON_PIN, INPUT_PULLUP);
    // Aseta LEDin alkuperäinen tila
    digitalWrite(LED_PIN, currentLedState);
    // Luo binaarinen semafori
    binarysemaphore = xSemaphoreCreateBinary();

    // Pysäytä ohjelman suoritus, jos semaforin luonti epäonnistuu
    if (binarysemaphore == NULL) {
        Serial.println("Semaforin luonti epäonnistui");
        while (1);
    }

    // Aseta keskeytys suoritettavaksi painonapin laskusreunalta
    attachInterrupt(digitalPinToInterrupt(BUTTON_PIN), onButton, FALLING);
```

Kuvio 35. Toimintojen alustukset

Tehtävien luonnissa olennaista on määritellä tehtävien prioriteetit oikein. Lopuksi ylimääräiset tehtävät poistetaan, ja kuviosta 36 voidaan todeta, että toistofunktio on tyhjä.

```
    // Luo tehtävät
    xTaskCreate(toggleLED,     // Luo tehtävä, joka vaihtaa LEDin tilaa
                "Toggle LED",  // Tehtävän nimi
                2048,          // Pino koko tavuina
                NULL,          // Parametrit (ei tässä käytössä)
                2,             // Tehtävän prioriteetti
                NULL);         // Tehtävän kahva (ei tässä käytössä)

    xTaskCreate(printSerial,   // Luo tehtävä, joka tulostaa LEDin tilan sarjaporttiin
                "Print Serial",// Tehtävän nimi
                2048,          // Pino koko tavuina
                NULL,          // Parametrit (ei tässä käytössä)
                1,             // Tehtävän prioriteetti
                NULL);         // Tehtävän kahva (ei tässä käytössä)

    vTaskDelete(NULL);  // Poista nykyinen tehtävä (setup-osa)
}

void loop() {
    // Pääsilmukka, joka suoritetaan toistuvasti (tässä ei käytetä)
}
```

Kuvio 36. Tehtävien luonti

3.9 Seurantaviikko 9: Zigbee-klusterikirjasto

Seurantaviikolla 9 työtehtäväni keskittyivät pääasiassa tukitehtäviin sekä vaihtolaitelistan purkamiseen välittämällä vaihtolaitteita palautuneiden tilalle. Kehitystyössä tutustuin edelleen Zigbee-verkon häiriötekijöihin Wi-Fi-verkkojen ja Bluetooth-laitteiden osalta sekä ehdin hieman tutustua Zigbee-klusterikirjastoon.

ZigBee Alliancen ylläpitämä Zigbee-klusterikirjasto (ZCL) on kokoelma klustereista, jotka määrittelevät, miten Zigbee-laitteet kommunikoivat keskenään. Klusterit ovat (ryhmiä, jotka sisältävät attribuutteja ja komentoja). ZCL on kirjasto, joka auttaa varmistamaan, että eri valmistajien Zigbee-laitteet voivat toimia yhdessä ja “puhua samaa kieltä”. (Digi International 2023.)

Klusterkirjaston tarkoitus on nopeuttaa Zigbee-laitteiden kehitystä ja standardisoida laitteiden välistä viestintää. Se jakaa laitteiden toiminnot clustereihin, jotka

määrittelevät yksittäisen toiminnallisuuden tai laiteominaisuuden. Tämä helpottaa laitteiden ohjelmointia ja varmistaa niiden yhteensopivuuden. (Elsevier 2008.)

Klusteri on kuin objekti: sillä on attribuutteja, dataa ja komentoja, kuten On/Off-kytkin (NXP 2008, 7). Attribuutit voivat olla esimerkiksi mittausarvoja, laitteen tilatietoja tai konfiguraatioasetuksia.

Klusterit on jaettu toiminnallisiin alueisiin eri tarkoitusten mukaan. Yleisiin toimintoihin kuuluvat esimerkiksi päälle- tai poiskytkeminen, tunnistautuminen ja tason säätö. Suljettavat rakenteet -alue kattaa clustereita, jotka on tarkoitettu ovi- ja ikkunajärjestelmiin. Lämmitys, ilmanvaihto ja jäähdytys (HVAC) -alue sisältää clustereita, jotka liittyvät lämpötilan ja ilmaston hallintaan. Valaistus-alue puolestaan sisältää valojen ohjaukseen liittyviä clustereita. (Digi International 2023.)

Klusterit on jaettu kahteen tyyppiin: serveriklusterit ja asiakasklusterit. Serveriklusterit tarjoavat tietoa tai toimintoja, kun taas asiakasklusterit käyttävät näitä toimintoja. (The ZigBee Alliance 2016, 46.)

Attribuutit määrittävät laitteen tilatiedot; esimerkiksi lämpötila-anturi voi sisältää attribuutin, joka tallentaa nykyisen lämpötilan arvon. Valolla voisi olla atribuuttina väri ja kirkkauden taso. (The ZigBee Alliance 2016, 40.)

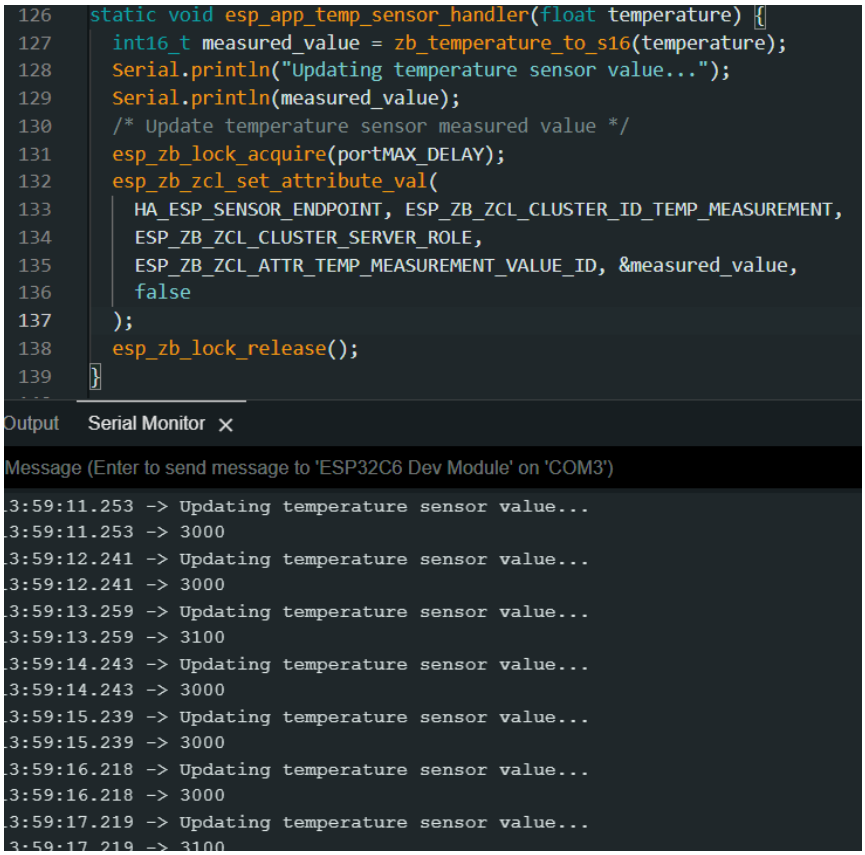
Komennot ovat kuin funktioita; niitä käytetään laitteen toiminnan ohjaamiseen tai tilamuutosten pyytämiseen. Esimerkiksi valokatkaisin voi lähettää komennon valaisimelle kytkeäkseen sen päälle tai pois päältä. (The ZigBee Alliance 2016, 40.)

3.10 Seurantaviikko 10: Zigbee-verkon muodostus

Seurantaviikolla 10 tehtäväni keskittyivät edelleen pääosin tukitehtäviin. Vaihtolaitteiden lähetystä ja palautuksien käsittelyä. Kehitystyön puolella ehdin tutustua kuitenkin keskusyksikköön. Tarkastelin ja testasin Linuxin crontabissa ajettavia skriptejä, jotka suoritetaan automaattisesti tietyin väliajoin. Nämä skriptit hoitavat

järjestelmän rutiininomaisia ylläpitotehtäviä ja varmistavat sen vakauden ja toimivuuden. Esimerkkeinä suoritettavista skripteistä ovat järjestelmän palvelujen valvonta ja automaattiset uudelleenkäynnistykset, jos jokin palvelu ei toimi odotetusti.

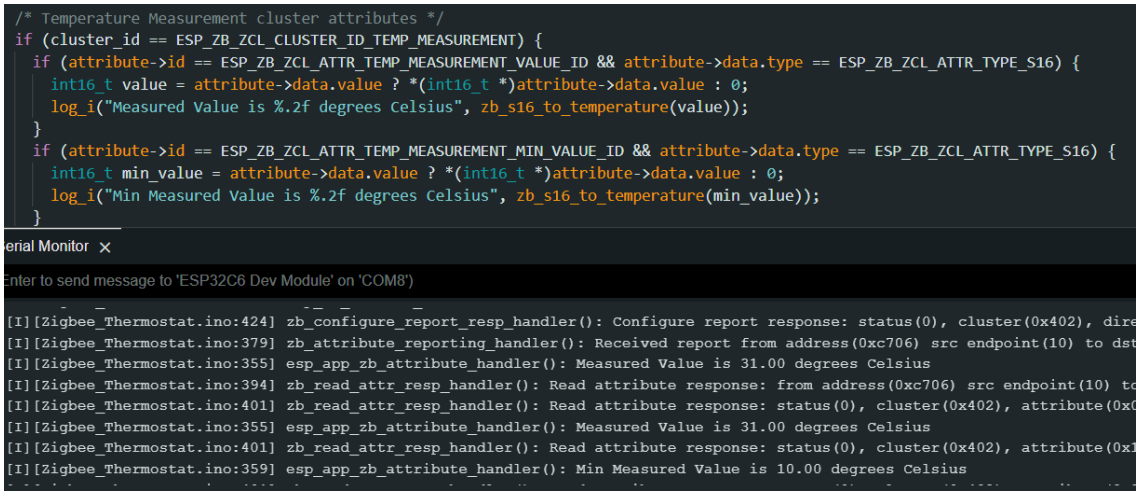
Tutustuin myös Espressifin tarjoamien mallien pohjalta ZigBeen toimintaan. Mallit hyödynsivät FreeRTOS-järjestelmää, joten aiemmasta RTOS-järjestelmään perehtymisestä oli tässä kohdin apua ohjelman toiminnan ymmärtämisessä. Käytössäni oli kaksi ESP32-C6-mikrokontrolleria, jotka tukivat ZigBee-yhteyksiä. Ensimmäinen ESP32 toimi end-devicenä, jossa oli lämpötilan luku- ja lähetysominaisuudet (kuvio 37).



Kuvio 37. End-device ja lämpötilan käsittelijä

Mallien pohjalta muodostin yksinkertaisen verkon, jossa end-device-laitteelta lähetettiin lämpötiladataa koordinaattorille, joka vastaanotti lämpötilatietoja tietyin väliajoin (kuvio 38). Näihin toimintoihin käytettiin edellisviikolla tutustuttua ZCL-kirjastoa, jonka avulla hyödynnettiin valmiita toimintoja ja päätepisteitä (*end*

points) lämpötilan mittaamiseen ja tiedonsiirtoon. Näiden mallien pohjalta on tarkoitus jatkossa tutkia mahdollisuuksia Zigbee-verkon optimointiin ja laajennettavuuteen, jotta Zigbee-verkot tulisivat syvällisemmin tutuksi.



Kuvio 38. Koordinaattorin vastaanottamat lämpötilat

4 POHDINTA

Opinnäytetyön tavoitteena oli auttaa minua kehittymään IoT-järjestelmäasiantuntijana sekä tutkia, kuinka IoT-järjestelmiä voidaan hyödyntää siten, että ikäihmiset voivat asua pidempään kotona. Lisäksi etsittiin keinoja, joilla IoT-järjestelmään perehdyttämistä voidaan helpottaa.

Opinnäytetyön toteutus auttoi minua ymmärtämään IoT-järjestelmiä paremmin. Jo tietoperustan laatiminen tuki suuresti järjestelmän kokonaisuuden hahmottamista, sillä sen luominen edellytti järjestelmän eri osa-alueiden ja niiden keskinäisten yhteyksien ymmärtämistä. Työn keskeisenä osana oli RTOS-käyttöjärjestelmä, joka oli minulle täysin uusi alue, ja siihen perehtyminen auttaa minua kehittämään IoT-järjestelmiä, koska RTOS-käyttöjärjestelmiä käytetään laajalti IoT-ympäristöissä.

RTOS-järjestelmä mahdollistaa järjestelmän reaktiivisen toiminnan, jolloin havainnot saadaan mahdollisimman reaaliaikaisesti. Lisäksi RTOS tukee vähävirtaista toimintaa, mikä on keskeistä IoT-laitteille. Vähävirtaisuutta edistää myös ZigBee-tiedonsiirtotekniikka, joka on kehitetty erityisesti vähävirtaisille laitteille. Vähävirtaisuuden ja langattoman tiedonsiirron ansiosta laitteet voivat olla huomaamattomia ja paristokäyttöisiä, mikä auttaa säilyttämään kodin viihtyisänä ja kotoisana.

IoT-järjestelmiin tutustumisen myötä olen ymmärtänyt, että järjestelmiä voidaan hyödyntää siten, että hoitohenkilökunta pystyy olemaan etäyhteydessä ikäihmisiin ja arvioimaan yksilöllistä tarvetta kotikäynneille ja hoidon tarpeelle. Seurannassa voidaan hyödyntää esimerkiksi tietoa aktiivisuudesta, ruokailusta, nukkumisesta ja lääkkeiden ottamisesta. Näin ollen ikäihmiset voivat asua kotonaan pidempään, kun heidän terveyttään ja toimintakykyään voidaan seurata etäältä.

IoT-järjestelmän perehdytyksen helpottamiseksi järjestelmä olisi hyvä dokumentoida huolellisesti erilaisin kaavioin, kuten sensoreiden aktiivisuuskaavioilla, jotka havainnollistavat sensorin toiminnan logiikkaa. IoT-järjestelmän kokonaisuudesta

olisi hyödyllistä luoda kerroksittainen arkkitehtuurikuvaus, joka auttaa ymmärtämään järjestelmän eri osa-alueita ja niiden välisiä yhteyksiä. Tämä selkeyttää järjestelmän kokonaisuuden hahmottamista.

Opinnäytetyötäni voidaan hyödyntää IoT-järjestelmän kokonaisuuden havainnollistamiseen ja sen käytön esittelemiseen tiedon keräämisessä. Työ voi myös tarjota kuvaa siitä, millaista työ järjestelmäasiantuntijan tehtävissä voisi olla.

Käytin työssäni tekoälyä kirjoitusten oikolukemiseen ja tekstin rakenteen muodostamiseen, missä se osoittautui erinomaiseksi avuksi. Tekoäly auttaa myös ymmärtämään järjestelmän eri osa-alueita, mutta tieto on varmistettava luotettavista lähteistä, joihin olen viitannut lähdeviitteillä.

Työn tekeminen oman palkkatyön ohella oli haastavaa, koska vapaa-aika täytyi käyttää tiedon hankintaan, asioiden ymmärtämiseen, toteuttamiseen ja kirjoittamiseen. Onnistuin kuitenkin motivoimaan itseäni tekemään työtä säännöllisesti. Viikoittainen raportointi auttoi tässä, sillä työn pystyi jakamaan viikoittain hallittaviin osiin. Opinnäytetyö auttoi minua tutustumaan IoT-järjestelmiin, koska jokaisella viikolla oli perehdyttävä uuteen aiheeseen. IoT-järjestelmä on laaja kokonaisuus, ja tulevaisuudessa aion käyttää enemmän aikaa eri osa-alueiden syvällisempään opiskeluun. Viikon aikana uudesta aiheesta ehtii työn ohella usein saada vain pintaraapaisun.

LÄHTEET

Amon, C., Kande, M., Lovett, A. & Ndabeni-Abrahams, S. 2020. The state of the connected world. World Economic Forum. Viitattu 30.8.2024 https://www3.weforum.org/docs/WEF_The_State_of_the_Connected_World_2020.pdf.

Antemijczuk, O., Berkolds, K., Chernov, V., Czekalski, P., Dautov, R., Distefano, S., Dobriborsci, D., Kapitonov, A., Kingsepp, M., Minnolo, A., Nikitenko, A., Pantiukhin, I., Paduch, J., Pietro, R., Puks, R., Rumba, R., Sell, R., Tokarz, K. & Vagale, A. 2019. Introduction to the IoT Coursebook in English. Viitattu 25.9.2024 https://iot-open.eu/download/iot-open-eu-introduction-to-the-iot-coursebook-in-english/.

Balbix 2024. Internet of Things (IoT) Biggest Security Challenges. Viitattu 3.10.2024 https://www.balbix.com/insights/addressing-iot-security-challenges/.

Bellehumeur, L. 2023. The 5 layers of IoT architecture that give it super powers. Viitattu 15.9.2024 https://novotech.com/blogs/news/the-5-layers-of-iot-architecture-that-give-it-super-power.

Biba, J. 2023. 14 types of IoT sensors available today. Viitattu 29.8.2024 https://builtin.com/articles/iot-sensors.

Castiglione, A., Umer, M., Sadiq, S., Obaidat, M. & Vijayakumar, P. 2021. The Role of Internet of Things to Control the Outbreak of COVID-19 Pandemic. Viitattu 1.9.2024 https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8769024/.

Digi International 2023. Clusters. Viitattu 16.11.2024 https://www.digi.com/resources/documentation/Digidocs/90002002/Concepts/c_zb_clusters.htm?TocPath=Zigbee+networks%7CZigbee+application+layers%3A+in+depth%7CApplication+profiles%7C_____1

Elsevier 2008. The ZigBee Cluster Library. Viitattu 16.11.2024 https://www.sciencedirect.com/science/article/abs/pii/B9780750685979000069.

Embetronicx 2023. RTOS Basics Concepts. Viitattu 15.10.2024 https://embetronicx.com/tutorials/rtos/freertos/rtos-basic-tutorial-for-beginners/.

Empirica 2024. Esineiden internet yksinkertaisesti selitettynä. Viitattu 2.9.2024 https://www.empirica.fi/iot.html.

Enisa 2019. Good Practices for Security of IoT. Viitattu 26.9.2024 https://www.enisa.europa.eu/publications/good-practices-for-security-of-iot-1.

Felch, R 2021. Understanding Zigbee and Wireless Mesh Networking. Viitattu 17.11.2024 https://www.blackhillsinfosec.com/understanding-zigbee-and-wireless-mesh-networking/.

Freertos 2024a. Binary semaphores. Viitattu 6.10.2024 https://www.freertos.org/Documentation/02-Kernel/02-Kernel-features/02-Queues-mutexes-and-semaphores/02-Binary-semaphores.

Freertos 2024b. Counting semaphores. Viitattu 6.10.2024 https://www.freertos.org/Documentation/02-Kernel/02-Kernel-features/02-Queues-mutexes-and-semaphores/03-Counting-semaphores.

Henderson, D. 2024. Zigbee Channels Explained. Viitattu 17.11.2024 https://www.smarthomeperfected.com/zigbee-channels/.

Hwan, H. & Kim, J. 2017. Proposal of Seamless Communication Method in Shadow Area Using LoRaWAN. Viitattu 17.11.2024 https://www.researchgate.net/publication/322248607_Proposal_of_Seamless_Communication_Method_in_Shadow_Area_Using_LoRaWAN.

Hymel, S. 2021. Introduction to RTOS. DigiKey Videot 20.9.2021. Viitattu 22.9.2024 https://youtu.be/playlist?list=PLEBQazB0HUyQ4hAPU1cJED6t3DU0h34bz.

KITRUM 2024. Three Layer Architecture in the Internet of Things. An Examination of Specifications and Security Threats. Viitattu 17.11.2024 https://kitrum.com/blog/three-layer-architecture-in-the-internet-of-things/.

Lahane, A. & Pawar, A. Design and Implementation of IoT based Smart Surveillance. Viitattu 17.11.2024 https://www.researchgate.net/publication/341870175_Design_and_Implementation_of_IoT_based_Smart_Surveillance.

Medium 2024. A Deep Dive into Heap, Stack, and Static Memory in Programming. Viitattu 29.9.2024 https://medium.com/@it.experts/a-deep-dive-into-heap-stack-and-static-memory-in-programming-3f22d9d1106d.

MQTT 2024. Why MQTT?. Viitattu 15.10.2024 https://mqtt.org/

NXP 2013. Co-existence of IEEE 802.15.4 at 2.4 GHz. Viitattu 27.10.2024 https://www.nxp.com/docs/en/application-note/JN-AN-1079.pdf.

NXP 2008. Freescale ZigBe Cluster Library (ZCL). Viitattu 16.11.2024 https://www.nxp.com/docs/en/reference-manual/ZCLRM.pdf.

ProCoders 2024. IoT Architecture: Definition, Key Layers, Challenges, and More. Viitattu 17.11.2024 https://procoders.tech/blog/iot-architecture/.

Simmons, A. 2022. Internet of things (IoT) architecture: layers explained. Viitattu 15.9.2024 https://dgtlinfra.com/internet-of-things-iot-architecture/.

Smartsight 2024. 5 IoT Layers for a perfect IoT Solution. Viitattu 15.9.2024 https://www.smartsight.in/technology/5-iot-architecture-layers-for-a-perfect-iot-solution/.

STMicroelectronics 2023. Introduction to LoRaWAN. Viitattu 2.10.2024 https://wiki.st.com/stm32mcu/wiki/Connectivity:Introduction_to_LoRaWAN.

Supermicro 2024. What is an IoT gateway? Viitattu 15.9.2024 https://www.supermicro.com/en/glossary/iot-gateway.

Suvanto Care Oy 2024a. Turvallisuusyritys. Viitattu 28.8.2024 https://www.suvantocare.fi/turvallisuusyritys/.

Suvanto Care Oy 2024b. Turvalaitteet. Viitattu 28.8.2024 https://www.suvantocare.fi/turvalaitteet/.

THL 2020. Kotona asumisen teknologiat ikäihmisille (KATI) 2021-2023. Ohjelma ja hankeopas. Helsinki: THL. Viitattu 28.8.2024 https://stm.fi/documents/1271139/2013549/KATI-ohjelma+ja+hankeopas+1.10.2020.pdf.

Wind River Systems 2024. What is a real-time operating system (RTOS)? Viitattu 15.9.2024 https://www.windriver.com/solutions/learning/rtos.

ZigBee Alliance 2016. ZigBee Cluster Library Specification. Viitattu 16.11.2024 https://zigbeealliance.org/wp-content/uploads/2019/12/07-5123-06-zigbee-cluster-library-specification.pdf.