Tampereen yliopisto

Joni Nieminen

REAALIAIKAISTEN KÄYTTÖJÄRJESTELMIEN AJANHALLINTA

Kandidaattitutkielma
Informaatioteknologian ja viestinnän tiedekunta
Tarkastaja: Sakari Lahti
Kesäkuu 2024

TIIVISTELMÄ

Joni Nieminen: Reaaliaikaisten käyttöjärjestelmien vertailua
Kandidaattitutkielma
Tampereen yliopisto
Sähkötekniikan kandidaattiohjelma
Kesäkuu 2024

Tämän työn tarkoituksena on tutustua käyttöjärjestelmien ajan hallintaan ja aikataulutuksen toteuttamiseen. Erityisesti tarkastellaan reaaliaikaisten käyttöjärjestelmien toteutusta. Tutkimuskysymyksinä ovat ”Miten käyttöjärjestelmä aikatauluttaa laitteiston laskentaresursseja?” ja ”Mitä eroja eri reaaliaikaisten käyttöjärjestelmien ajan hallinnan toteutuksilla on?”. Työssä käytetään eri käyttöjärjestelmien omia dokumentaatioita selvittämään, miten niiden toiminta on toteutettu. Lisäksi käytetään muita kirjallisuuslähteitä yleisen tiedon hankkimiseen.

Reaaliaikaiset käyttöjärjestelmät voidaan mieltää yleiskäyttöisten käyttöjärjestelmien laajentumina, joissa järjestelmän suorituskyky ei ole optimoinnin kohteena. Sen sijaan järjestelmän ennakoitavuus ja määräaikojen tarkkuus ovat optimoitavia ominaisuuksia. Reaaliaikaiset käyttöjärjestelmät eivät siis välttämättä ole nopeampia kuin yleiskäyttöiset käyttöjärjestelmät.

Työstä käy ilmi, että käyttöjärjestelmän tuottama laitteiston aikatauluttaminen voidaan tehdä monella eri tavalla. Työssä tutkitut kolme reaaliaikaista käyttöjärjestelmää: µC/OS-III, Eclipse ThreadX sekä freeRTOS käyttävät kuitenkin kaikki prioriteettiin pohjautuvaa etuoikeutettua aikataulutusta. Kaikkien kolmen aikataulutus on siis suurelta osin sama. Eclipse ThreadX mahdollistaa myös etuoikeuskynnyksen käyttämisen, mitä muut käyttöjärjestelmät eivät tue. Etuoikeuskynnyksellä tarkoitetaan prosessin kykyä olla tulematta ohitetuksi.

Avainsanat: Reaaliaikainen käyttöjärjestelmä, RTOS, aikataulutus, ajan hallinta, µC/OS-III, Eclipse ThreadX, freeRTOS

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

FAT	Tilanvaraustaulukko (engl. *File Allocation Table*)
FIFO	Jono (engl. *First in, First out*)
FPGA	Uudelleenohjelmoitava porttimatriisi (engl. *Field Programmable Gate Array*)
GPOS	Yleiskäyttöinen käyttöjärjestelmä (engl. *General Purpose Operating system*)
HW	Laitteisto (engl. *Hardware*)
ISA	Käskykanta-arkkitehtuuri (engl. *Instruction Set Architecture*)
ISR	Keskeytyspalvelurutiini (engl. *Interrupt service routine*)
PCB	Prosessielementti (engl. *Process Control Block*)
PSD	Prosessin tilakaavio (engl. *Process state diagram*)
RTOS	Reaaliaikainen käyttöjärjestelmä (engl. *Real Time Operating System*)
RTS	Reaaliaikajärjestelmä (engl. *Real-Time System*)
SW	Ohjelmisto (engl. *Software*)
WCET	Pahimman tapauksen suoritusaika (engl. *Worst Case Execution Time*)

.

1. JOHDANTO

Reaaliaikaisten järjestelmien (engl. Real-time system, RTS) toiminnan kannalta aika on kriittinen tekijä. Niiden oikeaan käyttäytymiseen tarvitaan oikea toiminnallisuus sekä oikea ajoitus. Esimerkiksi auton ilmatyynyn toiminnallisuus vaatii, että tyyny täyttyy kaasulla riittävän lyhyessä ajassa. Oikeaan ajoitukseen vaaditaan se, ettei turvatyyny laukea liian myöhään eikä liian ajoissa. Huomataan, että kokonaisuudessa turvatyynyn tai minkä tahansa muun reaaliaikaisen järjestelmän käyttäytyminen vaatii sekä oikean lopputuloksen, että ajoituksen [1], [2].

Tyypillisesti sulautetuissa järjestelmissä käsitellään järjestelmän ulkopuolista aikatasoa. Järjestelmän täytyy siis kyetä reagoimaan reaalimaailman tapahtumiin. Monet sulautetut järjestelmät ovat täten reaaliaikaisia järjestelmiä [3]. Jotta reaaliaikaisen järjestelmän laajenevaa kompleksisuutta voidaan hallita, tarvitaan reaaliaikaisia käyttöjärjestelmiä.

Reaaliaikaiset käyttöjärjestelmät (engl. Real time operating system, RTOS) voi mieltää yleiskäyttöisten käyttöjärjestelmien (engl. General purpose operating system, GPOS) laajentumina. Tässä työssä ensin tarkastellaan yleiskäyttöisiä käyttöjärjestelmiä ja sitten tutkitaan, mitä lisäominaisuuksia reaaliaikaisilla käyttöjärjestelmillä saavutetaan. Yleiskäyttöiset käyttöjärjestelmät, kuten Windows tai Android, on suunniteltu käyttäjäystävällisiksi, eli näiden tarkoituksena on saada käyttäjälle mahdollisimman saumaton käyttökokemus. Tämä tarkoittaa sitä, että kun käyttäjän syötteisiin reagoidaan, muu toiminnallisuus saattaa viivästyä. RTOS puolestaan antaa varmuuden ajanhallintaan, sillä järjestelmän toiminnan täytyy olla ennakoitavissa. Tämä ennakoitavuus tarkoittaa, että käyttöjärjestelmä pystyy takaamaan määräajoissa pysymisen. Vastaavasti GPOS pyrkii maksimoimaan toteutuneen laskennan määrän, eli järjestelmän suorituskyky on optimoitu. [4]

Tässä tutkimuksessa vertaillaan kolmea eri reaaliaikakäyttöjärjestelmää: µC/OS–III, Eclipse ThreadX sekä freeRTOS. Nämä ovat valittu vertailtaviksi, koska niistä ThreadX ja freeRTOS ovat ilmaiseksi saatavilla. µC/OS-III puolestaan on maksullinen, mutta hyvin suosittu RTOS-järjestelmä. Tutkittavana ominaisuutena on ajan hallinta ja aikataulutus. Lisäksi sivutaan käyttöjärjestelmien tarvitseman muistin vaatimuksia.

Ensin luvussa 2 tutustutaan käyttöjärjestelmiin, yleiskäyttöisiin sekä reaaliaikaisiin, sekä pohditaan niiden eroja ajan hallintaa painottaen. Lisäksi luvussa kaksi paneudutaan tarkemmin reaaliaikaisten käyttöjärjestelmien ytimien toimintaan ja ytimen toiminnallisuuksien toteutukseen. Luvussa 3 tutustutaan kolmen eri RTOS-järjestelmän toteuttamaan ajanhallintaan. Luvussa 4 analysoidaan vertailun tulokset. Lopuksi luvussa 5 kootaan tulokset yhteen ja pohditaan, miten tutkimusta voitaisiin jatkaa.

2. KÄYTTÖJÄRJESTELMÄT

Käyttöjärjestelmiä on erilaisia. Arkielämästä tutut Microsoft Windows tai Android ovat monilla jokapäiväisessä käytössä. Kuitenkin käyttöjärjestelmien tekninen toteutus on monesti muutakin kuin mitä arkielämässä nähtävä ikkunointi antaa ymmärtää. Käyttöjärjestelmän tarkoitus on ohjata tietokoneen fyysistä toteutusta eli laitteistoa (engl. Hardware, HW) ohjelmallisesti (engl. Software, SW) [5].

Hierarkkinen tietokone, jollaisen Haikala esittää, on kone, jonka laitteiston ja sovelluskerroksen välillä on käyttöjärjestelmä. Tämä kuvaa käyttöjärjestelmän sijainnin tietokoneen rakenteessa. Käyttöjärjestelmän pääasiallinen tarkoitus on ohjata eri sovellusten pääsyä laitteiston resursseille ja taata, että laitteiston suorittimella, prosessorilla, ei ole liikaa käyttäjiä samaan aikaan. [5]

Suorituksessa olevaa ohjelmaa kutsutaan prosessiksi [5], tässä työssä prosessi ja tehtävä tarkoittavat samaa asiaa, kuitenkin työssä käytetään selvyyden vuoksi pääosin termiä prosessi. Tällaisia prosesseja voidaan aikatauluttaa, eli niiden suoritusjärjestystä muuttaa. Tässä työssä tarkastellaan nimenomaan prosessien aikatauluttamista ja aikataulutuksen toteuttamista eri RTOS- järjestelmillä.

Lisäksi käyttöjärjestelmä vastaa muistinhallinnasta laitteistotasolla. Muistinhallinta on kriittinen osa kaikkia tietokoneita, sillä ilman muistinhallintaa prosesseja ei voitaisi keskeyttää eikä jaotella. Käyttöjärjestelmän vastuu on huolehtia tarvittavien tietojen tallentamisesta muistiin. Kun prosessi keskeytetään, suoritetaan ympäristön vaihto (engl. context switch) [6], [7], [8]. Myöskin yleinen informaation säilöminen muistiin ja muistista haku tapahtuu käyttöjärjestelmän välityksellä [9].

Koska prosessorilla voidaan toteuttaa rajallinen määrä laskentaa, käyttöjärjestelmän perusidea on antaa järjestelmän sisäisille rakenteille vuoronsa käyttää suorittimen laskentapalvelua. Näin varmistutaan, että laitteistotason suunnittelussa voidaan keskittyä bittimanipulaatioon ja laskentaan eikä tarvitse huolehtia ajonaikaisesta aikataulutuksesta. Kaikki tämä voidaan toteuttaa ohjelmallisesti, mikä helpottaa siirrettävyyttä, laajennettavuutta ja käytettävyyttä.

2.1 Yleiskäyttöiset käyttöjärjestelmät

Yleiskäyttöiset käyttöjärjestelmät (GPOS) ovat monille tuttuja arkielämästä. Microsoft Windows ja macOS ovat yleisimmät tietokoneissa käytettävät käyttöjärjestelmät. Vastaavasti Android ja iOS ovat yleisimmät matkapuhelimissa käytettävät käyttöjärjestelmät.

Käyttöjärjestelmien yksi tarkoitus on helpottaa tietokoneille, tai puhelimille, uusien ohjelmistojen kehittämistä. Niiden avulla voidaan luottaa sovelluskerroksen suunnittelussa, että laitteiston suorittimen ajoitus on kunnossa ja muistin suojaus toimii. Niinpä voidaan sovellustason ohjelmien, ohjelmistojen, suunnittelussa keskittyä sovelluksen toiminnallisuuden toteuttamiseen. [5]

Lisäksi käyttöjärjestelmien käytössä merkittävä etu on saman ohjelmiston ajaminen eri laitteilla, joilla voi olla erilainen laitteisto, olettaen laitteiston olevan yhteensopiva käyttöjärjestelmän kanssa. Siispä ohjelmiston voi onnistua ajaa myös vanhemmalla laitteella, jos käyttöjärjestelmä voi käyttää laitteistoa.

2.2 Reaaliaikaiset käyttöjärjestelmät

Yleiskäyttöiset käyttöjärjestelmät ovat toiminnaltaan lähellä reaaliaikaisia käyttöjärjestelmiä, molemmissa tarkoituksena on saada suorittimen laskentaresurssit jaettua monien sovelluksien välillä ja tuottaa muistinhallintaa. RTOS-järjestelmissä on kuitenkin tärkeä ero ajoituksen kannalta, sillä RTOS-järjestelmillä vaatimuksena on saada suorittimen vasteajat ennakoitua [10].

RTOS on erityisesti käytössä sulautetuissa järjestelmissä, missä ajoitus on kriittinen osa kokonaisuuden toimivuuden kannalta. Yleiskäyttöisellä käyttöjärjestelmällä voi syntyä tilanteita, jossa jokin sovellus nälkiintyy, siis se ei saa prosessorilta laskentaresursseja, sillä jokin muu sovellus menee nälkiintyvän käyttäjän edelle prioriteetissa. Myös lukkiutuminen, jossa kaikki sovellukset ovat odotustilassa, voi tapahtua GPOS-järjestelmissä. [5]

Ongelmia voidaan ratkaista toki myös GPOS:n yhteydessä, mutta RTOS tuo valmiin ratkaisun näihin ongelmiin. Lisäksi RTOS-järjestelmässä ratkaisun täytyy olla riittävän nopea aikarajoitteiden mukaan.

RTOS-järjestelmien suosio on noussut paljon viime vuosien aikana. On kehitetty uusia käyttökohteita kuten miehittämättömät lennokit ja ajoneuvot, jotka vaativat toimiakseen

järjestelmän, joka kykenee reagoimaan reaalimaailman tapahtumiin reaaliaikaisesti. RTOS mahdollistaa tällaisten tuotteiden toteuttamisen.

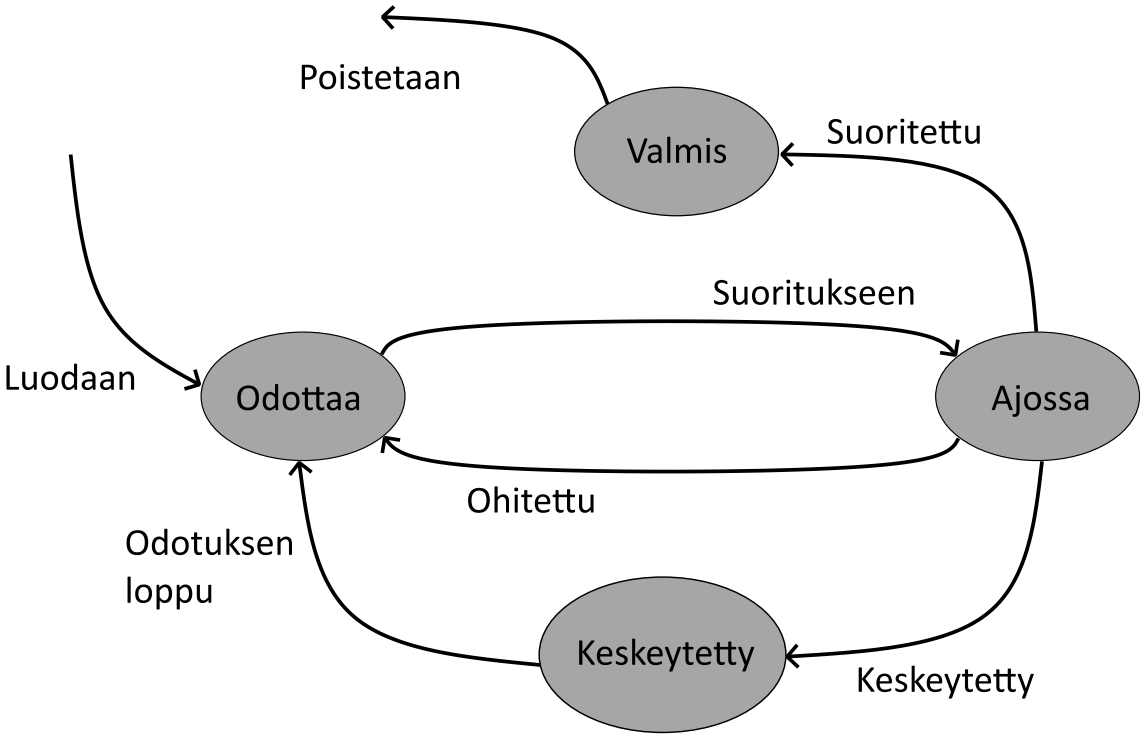
2.3 Ydin

Käyttöjärjestelmän ytimellä on useita tehtäviä, tässä käsitellään tarkemmin vain kahta: aikataulutusta sekä laitteiston hallintaa keskeytysten ja aikataulutuksen yhteistuloksena. Muistinhallintaa sivutaan myös hieman. Lisäksi ydin hoitaa muun muassa oheislaitteiden käsittelyä ja niihin liittyvää ohjausta. [5]

Ytimen pääasiallinen tehtävä on huolehtia prosesseista. Prosessi on ohjelma, joka voidaan käyttöjärjestelmän avulla suorittaa. Prosessi puolestaan koostuu säikeistä, jotka ovat käyttöjärjestelmän kannalta perusyksiköitä. Näitä säikeitä voidaan aikatauluttaa ja siten prosessit voidaan suorittaa järjestyksessä. Ytimessä tätä tehtävää hoitaa prosessielementti (PCB, Process control block). [5]

2.3.1 Prosessit

Prosessilla tarkoitetaan tässä työssä ohjelmaa, joka suoritetaan laitteiston prosessorilla, suorittimella. Prosessi, joka koostuu säikeistä, käsitetään tästä lähtien yhtenä yksikkönä, joita käyttöjärjestelmä käsittelee. Käyttöjärjestelmä aikatauluttaa prosessit ja antaa niille laitteiston laskentaresurssit käyttöön. Prosessit voidaan jakaa eri tiloihin: odottaa, ajossa, keskeytetty ja valmis, tiloja vastaava tilakaavio esitetään kuvassa 1.



Kuva 1. Prosessin tilakaavio.

Kuten kuvasta näkyy, prosessit luodaan tilaan ”odottaa”. Tässä tilassa prosessi on jonossa ja odottaa pääsyä prosessorille. Kun käyttöjärjestelmän aikataulutus antaa luvan suoritukseen, prosessi siirtyy tilaan ”ajossa”. Tällöin prosessorilla suoritetaan kyseistä prosessia, tämä voidaan kuitenkin käyttöjärjestelmän aikataulutuksen toimesta tulla ohitetuksi. Tällöin korkeamman prioriteetin ohjelma, jolla on etuoikeus, ohittaa prosessin ja prosessi siirtyy takaisin tilaan ”odottaa”. ”Ajossa”-tilasta prosessi voi myös keskeytyä, esimerkiksi odottaa oheislaitteen lukua. Tällöin prosessin tila muuttuu ”keskeytetty”-tilaan, josta odotuksen jälkeen se siirtyy takaisin ”odottaa”-tilaan, jossa se aikataulutetaan. Lopulta, kun prosessi on saatu suoritettua ajon jälkeen, prosessin tila on ”valmis”. Tästä tilasta prosessi poistetaan. Prosessien luominen ja poistaminen jätetään tämän työn ulkopuolelle.

Prosesseilla on konteksti, eli ympäristö, joka pitää sisällään prosessorin sen aikaisen tilan, käytännössä rekisterien arvot ja ohjelmalaskuri [11]. Tämä ympäristön vaihto (engl. context switch) on epäedullista, sillä tänä aikana prosessoria ei käytetä [12]. Ympäristön vaihto on yksi merkittävin prosessorin aikatehokkuutta heikentävä tekijä [11].

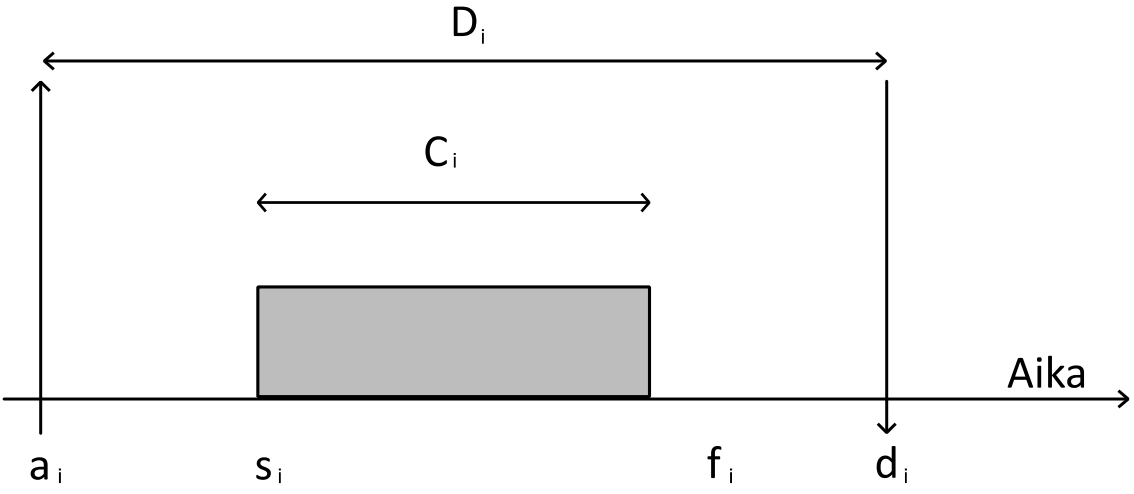
2.3.2 Aikataulutus ja ajan hallinta

Aikataulutus eli skedulointi eli vuoronnus on menetelmä, jolla selvitetään seuraava tai seuraavat suoritettavat prosessit. Tätä tehtävää hoitaa aikatauluttaja eli skeduloija eli

vuorontaja. Suoritettavissa olevat prosessit asetetaan tärkeysjärjestykseen, aikatauluun, ja ensimmäisenä suoritetaan tärkein. Tyypillisesti aikataulutuksen tavoite on optimoida järjestelmän suorituskyky, mutta reaaliaikaisissa järjestelmissä optimoinnin kohteena on vasteaikojen ennustettavuus [5]. Aikataulun järjestys on tyypillisesti prioriteettiin pohjautuva.

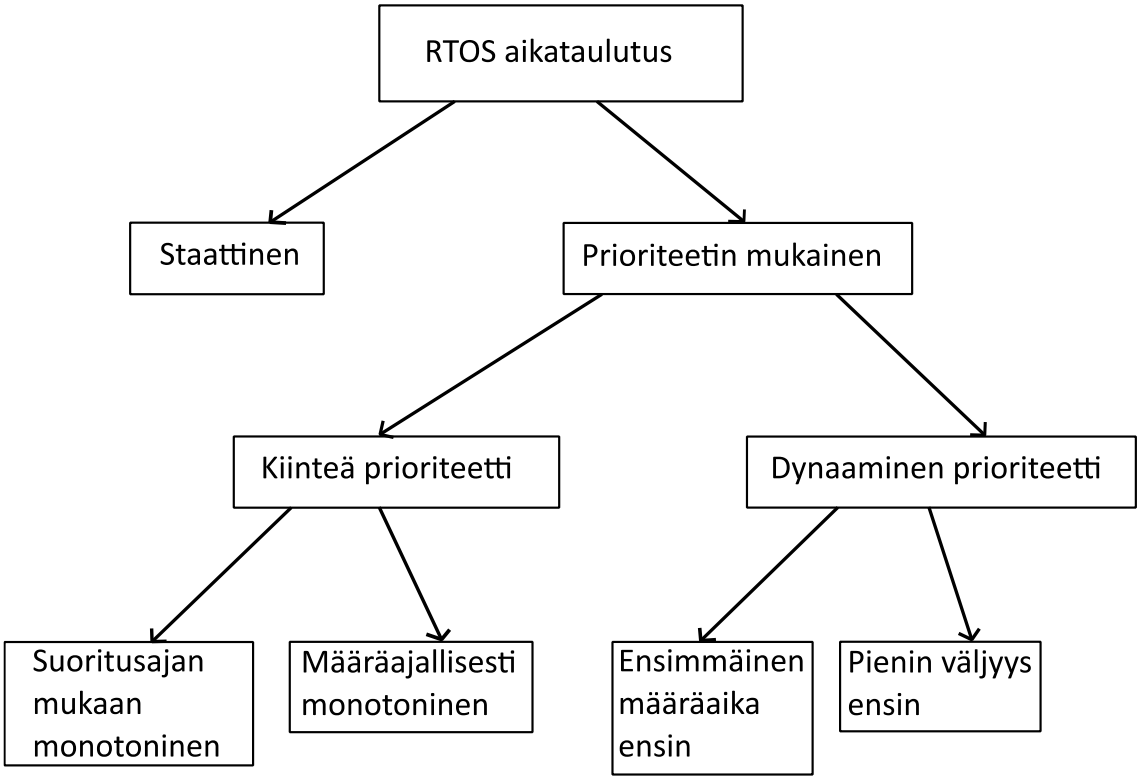
Keskeytykset ovat merkittävä osa aikataulutusta, sillä keskeytyksen tultua käyttöjärjestelmän täytyy reagoida keskeytykseen ja sen aiheuttajaan. Yleisesti käyttöjärjestelmät voivat keskeytyksen saatuaan muuttaa suoritettavaa prosessia dynaamisesti, jolloin keskeytys saadaan heti hoidettua. Tämä ei useinkaan ole mahdollista reaaliaikaisen järjestelmän tapauksessa, sillä prosessien aikatauluttaminen keskeytysten vaikutuksesta voi johtaa määräämättömiin viiveisiin [10].

Tarkastellaan tarkemmin käyttöjärjestelmien erityisesti RTOS-järjestelmien ajan hallintaa ja miten prosessien ajoitusta voidaan tutkia. Olkoon prosessi J_i reaaliaikainen prosessi, jonka saapumisaika, eli aika, jolloin prosessi on valmis suoritukseen, on a_i. Prosessin pahimman tapauksen suoritusaika, eli pisin mahdollinen suoritukseen kuluva aika WCET-aika (engl. Worst Case Execution Time), on C_i. Absoluuttinen määräaika D_i, eli aika, jolloin prosessi täytyy olla suoritettuna järjestelmän ulkopuolisen ajan suhteen. Suhteellinen määräaika d_i, joka kuvaa ajan, jolloin prosessi tulee olla suoritettuna verrattuna saapumisaikaan. Aloitusaika s_i, on aika, jolloin prosessin suoritus aloitetaan. Lopetusaika f_i, eli aika, jolloin prosessi on saatu suoritettua. [10] Näistä C_i vähenee sitä mukaan, kun prosessin suoritus etenee [13]. Ajoituskaavio, kuva 2, esittää reaaliaikaisen prosessin J_i ajoituksen.



Kuva 2. Reaaliaikaisen prosessin J_i ajoituskaavio, muokattu lähteestä [10].

Erilaisia aikataulutuksen toteuttavia algoritmimuotoja on esitetty kuvassa 3. Näistä staattinen aikataulutus tarkoittaa aikataulutusta, joka on määrätty jo käyttöjärjestelmän käännösvaiheessa [14]. Siis suoritettavien prosessien järjestykseen ei voi mitenkään vaikuttaa enää ajon aikana.



Kuva 3. RTOS aikataulutuksen luokittelu, muokattu lähteestä [14].

Kiinteän prioriteetin menetelmä on selkeämpi kahdesta prioriteetin mukaisesta menetelmästä. Kiinteällä prioriteetilla tarkoitetaan prioriteettia, joka prosessille on annettu etukäteen, eikä sitä voi muuttaa. Suoritusajan mukainen priorisointi tarkoittaa sitä, että prosessin prioriteetti on sitä suurempi, mitä lyhyempi prosessin suorittamiseen kuluva aika on. Siis nopeimmat prosessit suoritetaan ensin. Määräajallisesti priorisointi taas antaa prosesseille prioriteetin, sen mukaan, mikä on prosessin suorituksen määräaika. Siis kriittisin prosessi suoritetaan ensin. [14]

Dynaamisen prioriteetin menetelmä on joustavampi suorituksen aikana, sillä prioriteetit voidaan määritellä ajon aikana uudestaan. Kuitenkin yleensä jokin prioriteetti on asetettu jo ennestään. Ensimmäinen määräaika ensin tarkoittaa sitä, että seuraavaksi suoritetaan

se prosessi, jonka määräaika on lähinnä. [14] Väljyys lasketaan jokaiselle prosessille vähentämällä määräajasta jäljellä oleva suoritettava aika kaavalla $L_i = D_i - C_i$. Se prosessi, jonka väljyys on pienin, eli on lähimpänä ylittämässä määräaikansa, suoritetaan ensin. Suorituksen aikana väljyys ei muutu. [13]

Ilman prioriteettejakin voi luoda pieniä järjestelmiä, mutta suuremmissa järjestelmissä ilman prioriteetteja aikataulutuksen toteuttaminen prosessien sisäisten rakenteiden avulla on vaikeaa. Siispä prioriteetit helpottavat aikataulusta ja saavat sen helpommin ymmärrettäväksi ja laajennettavaksi.

2.3.3 Keskeytyspalvelurutiini

Jotta prioriteetteja voidaan käyttää, täytyy aikataulutuksen tietää, milloin uusi prosessi tulee jonottamaan suoritusta. Keskeytykset Ilmoittavat järjestelmälle, että jokin reagointia vaativa toimenpide on tullut suoritettavaksi. Keskeytysten käsittely toimii melko yksinkertaisesti: tallennetaan ympäristö (engl. context) ja haaraudutaan keskeytyskäsittelijään, tässä tapauksessa keskeytyspalvelurutiinin avulla [5].

Keskeytyksen alussa tehtävä ympäristön vaihto vaatii suorittimen rekisterin tallettamisen, mikä voidaan tehdä ohjelmallisesti sekä ohjelmalaskurin ja tilarekisterin tallettamisen, mikä täytyy tehdä laitteistotasolla [5]. Keskeytyksen aiheuttaneen tapahtuman suorituksen jälkeen käyttöjärjestelmä vastaa aikataulutuksesta. Jos keskeytyksen käsittelyn ja suorituksen aikana on tullut muitakin korkean prioriteetin prosesseja jonoon, ne suoritetaan ennen kuin palataan keskeytystä edeltävään tilaan. Kuitenkin aina palataan takaisin siihen prosessorin tilaan, jossa keskeytys sai aikaan muutoksen.

Keskeytyspalvelurutiini huolehtii myös sisäkkäisistä keskeytyksistä. Jos keskeytyksen suorituksen aikana tulee uusi keskeytys, jonka prioriteetti on korkeampi, vaihdetaan suoritettavaa prosessia uudestaan. Tämä on usein valinnainen ominaisuus ja yksinkertaisempi toteutus on lakkauttaa kaikki keskeytykset, kun yhtä keskeytystä suoritetaan.

2.3.4 Muistin hallinta

Prosessien aikataulutus vaatii muistia. Prosessien keskinäinen suoritusjärjestys eli aikataulu täytyy tallettaa jonnekin. Lisäksi prosessien ympäristö täytyy tallentaa, jotta prosessien vaihtaminen onnistuu. Jos prosessi keskeytyy, koska korkeamman prioriteetin

prosessi tulee jonoon suoritettavaksi, täytyy ohitettavan prosessin ympäristö tallentaa. Korkean prioriteetin prosessin suorittamisen jälkeen palataan takaisin täsmälleen samaan kohtaan, mistä prosessin vaihto tapahtui [5].

Koska keskeytyksiä voi olla useita sisäkkäin, täytyy riittävä muistin määrä taata ylivuodon estämiseksi. Tämän vuoksi tulee olla myös selvyys siitä, kuinka paljon muistia yksi keskeytyksen taso vaatii ja kuinka monta keskeytystasoa voi olla.

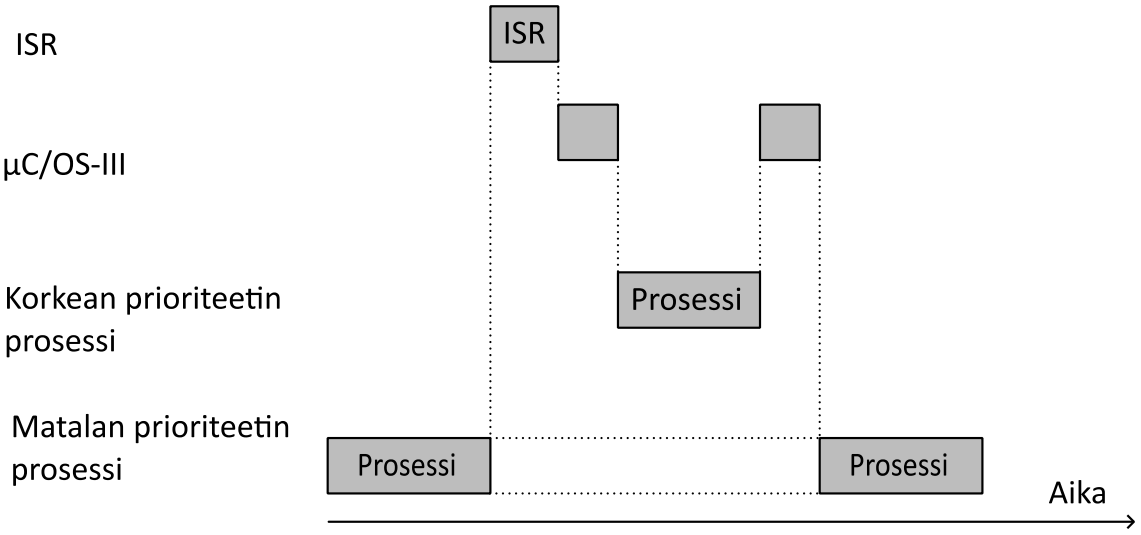
3. VERTAILTAVIEN KÄYTTÖJÄRJESTELMIEN AJAN HALLINTA

Tässä työssä käsitellään kolmea eri reaaliaikaista käyttöjärjestelmää: µC/OS-III, Eclipse ThreadX sekä freeRTOS. µC/OS-III on alkujaan vuonna 2009 julkaistu käyttöjärjestelmä. Se on maksullinen, jonka käyttö elektroniikkatuotteessa vaatii lisenssin. Eclipse ThreadX on Microsoftin kehittelemä reaaliaikainen käyttöjärjestelmä, jonka lähdekoodi on ilmaiseksi ladattavissa. FreeRTOS on MIT:n (Massachusetts Institute of Technology) avoimeen lähdekoodiin perustuva reaaliaikainen käyttöjärjestelmä. FreeRTOS on alkujaan Richard Barryn vuonna 2003 kehittämä käyttöjärjestelmä, josta on eri versioita. FreeRTOS on myös ilmaiseksi ladattava, mutta muut versiot, openRTOS ja safeRTOS, ovat maksullisia.

3.1 µC/OS-III

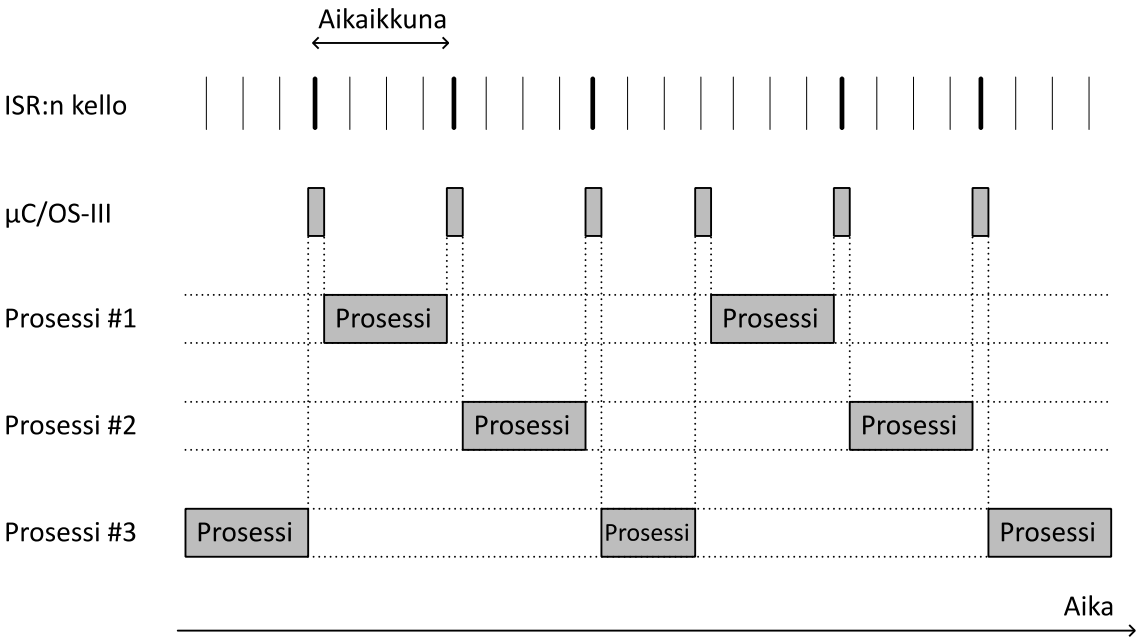
µC/OS-III:n aikataulutus mahdollistaa monen eri prosessin suorituksen, kunhan laitteistolla on käytettävissä riittävästi muistia. Siis ohjelmallisesti ei ole rajoitettu prosessien määrä, joka voidaan kerralla aikatauluttaa. [6] Pohditaan sitten, miten aikataulutus voidaan suorittaa, jotta nämä eri prosessit pääsevät reaaliaikaisuuden vaatimiin aikarajoitteisiin.

Käytössä on prioriteettiin perustuva etuoikeutettu aikataulutus, jossa kullekin prosessille määrätään prioriteetti sovelluksen mukaan. Laitteiston laskentaresurssin saa käyttöönsä puolestaan se prosessi, joka on valmis suoritettavaksi ja jolla on korkein prioriteetti. Etuoikeutettu aikataulutus puolestaan viittaa siihen, että jo suorituksessa oleva prosessi joutuu väistämään korkeamman prioriteetin edeltä. [6] Tätä on esitetty kuvassa 4



Kuva 4. Prioriteetin mukaan korkeamman prosessin suorittaminen ennen matalamman prioriteetin prosessia. Muokattu lähteestä [6].

On myös mahdollista, että kaksi tai useampia samalla prioriteetilla olevaa prosessia on valmiina suoritettavaksi. Tällöin käytetään kiertovuorottelua (engl. Round robin scheduling), jossa kullekin prosessille annetaan tietty aikaikkuna. Sitten tämän ajan jälkeen vuoro vaihtuu seuraavalle prosessille. Aikaikkunan saanut prosessi voi kuitenkin vapauttaa ajan toisten käyttöön, jos kyseinen prosessi ei tarvitsekaan koko aikaikkunaa [6]. Tätä on esitetty kuvassa 5.



Kuva 5. Kiertovuorottelun ajoitus, kun kaikilla prosesseilla on sama prioriteetti. Muokattu lähteestä [6].

Kuvassa 5 on hyvä huomata kohta, jolloin prosessi #3 suoritetaan toiseen kertaan. Silloin kyseinen prosessi ei käytä koko aikaikkunaa, mikä sille on varattu. Kuitenkin prosessi #1 saa käyttöönsä kokonaisen aikaikkunan, neljä kellojaksoa. Siis käyttöjärjestelmän huolehtimana aikaikkuna nollautuu, kun prosessia vaihdetaan.

3.2 Eclipse ThreadX

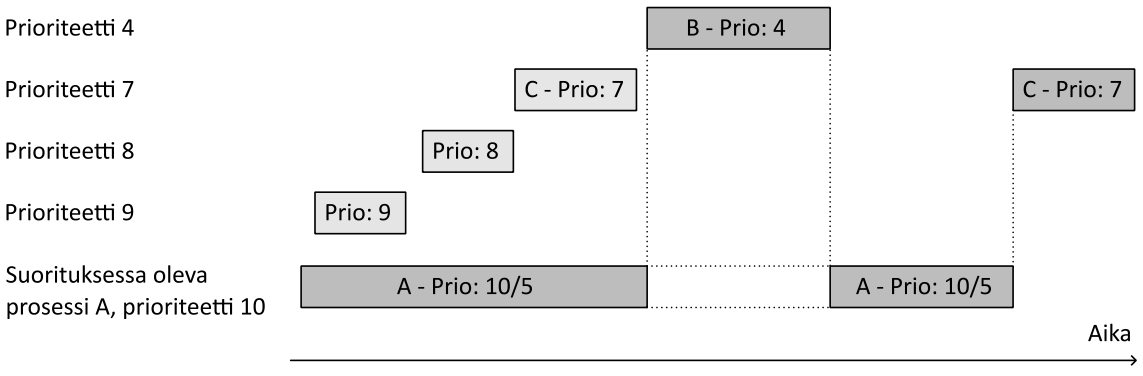
Myös Eclipse ThreadX käyttää prioriteettiin pohjautuvaa etuoikeutettua aikataulutusta. Oletuksena ThreadX käyttää 32 eri prioriteettitasoa, tämä määrä voidaan kuitenkin käyttäjän toimesta muuttaa. Eri prosesseilla voi myös olla sama prioriteetti. [7] Saman prioriteetin omaavat prosessit voidaan suorittaa kahdella tavalla, kiertovuottelulla tai FIFO-periaatteella.

Kiertovuorottelu toimii samaan tapaan kuin µC/OS-III:n kiertovuorottelu, jossa prosesseille annetaan aikaikkuna. Jos prosessi ei tule valmiiksi tässä aikaikkunassa, se joutuu odottamaan uudestaan, kunnes sille annetaan uusi aikaikkuna. Jos kiertovuorottelun aikana suoritettavaksi tulee korkeamman prioriteetin prosessi, suorituksessa oleva prosessi keskeytyy. Tämä keskeytetty prosessi jatkaa suoritusta, kun korkeamman prioriteetin prosessi on saatu suoritettua. Kuitenkaan sen aikaikkuna ei nollaudu, eli se ei saa lisää prosessorin suoritusaikaa [7].

FIFO-periaatteella (engl. First-In, first-out), eli jonoperiaatteella puolestaan suorittamista odottavat prosessit kerätään jonoon saapumisjärjestyksen mukaan. Suoritukseen pääsee se prosessi, joka on jonossa ensimmäisenä. Näin jokainen prosessi saa oman vuoronsa prosessorin laskentaresursseille ja suorittaa tehtävänsä, jonka jälkeen seuraava prosessi tulee suoritukseen, kunnes jono on tyhjä. Tämän jälkeen voidaan suorittaa alemman prioriteetin prosesseja.

Käyttäjän opas [7] ohjeistaa olemaan käyttämättä kiertovuorottelua aikaikkunoiden takia, jos suoritusta odottavilla prosesseilla on kullakin eri prioriteetti. Tällöinhän prosessit suoritetaan prioriteetin mukaisessa järjestyksessä eikä niitä tarvitse pilkkoa osiin. Aikaikkunoiden käyttö puolestaan lisää yleisrasitetta (engl. Overhead) hieman [7]. Tämä puolestaan hieman hidastaa järjestelmän toimintaa, joten aikaikkunoita ei kannata käyttää turhaan.

Etuoikeutettu aikataulutus ThreadX:n tapauksessa on melko sama kuin µC/OS-III:n järjestelmässä. Kuitenkin ThreadX sisältää yhden lisäominaisuuden: etuoikeuskynnyksen. Etuoikeuskynnyksen käyttö tarkoittaa sitä, että prosessin etuoikeus vaatii riittävän suuren prioriteettieron [7]. Esimerkiksi olkoon etuoikeuskynnys 5. Prosessi, jonka prioriteetti on 10 ei voi tulla ohitetuksi prioriteetin 9, 8, 7, 6 tai 5 omaavan prosessin toimesta. Prosessilla on ikään kuin kaksi prioriteettia, ensimmäinen prioriteetti kuvastaa prosessin kykyä ohittaa muita prosesseja. Toinen prioriteetti kuvastaa prosessin kykyä estää muiden prosessien väistämistä, siis ikään kuin suoritusajan prioriteetti; kun prosessi on suorituksessa, mikä täytyy olla toisen prosessin prioriteetti, jotta se voi ohittaa suorituksessa olevan prosessin. [15] Havainnollistetaan esimerkkiä kuvassa 6.



Kuva 6. Etuoikeuskynnyksen toiminta prosessien aikataulutuksessa.

Kuvasta 6 voidaan nyt huomata, että prosessi A, jonka alkuperäinen prioriteetti on 10 ei tule ohitetuksi, kunnes prosessi B, jonka prioriteetti ylittää kynnyksen, lopulta ohittaa prosessin A. Siis vaalealla taustalla olevat prosessit eivät saa etuoikeutta prosessorin laskentaresursseille, koska prosessin A suoritusaikainen prioriteetti on 5.

Huomattavaa on, että kun korkean prioriteetin prosessi B on saatu suoritettua, palataan täsmälleen siihen tilanteeseen, mikä vallitsi ennen kuin prosessi B valittiin suoritettavaksi. Tämän vuoksi palataan prosessiin A, vaikka korkeamman prioriteetin prosesseja onkin odottamassa suoritusta. Vasta, kun prosessi A saadaan suoritettua, suoritetaan prioriteetiltaan korkein prosessi C normaalin aikataulutuksen mukaisesti.

3.3 FreeRTOS

FreeRTOS käyttää samaa prioriteettiin pohjautuvaa etuoikeutettua aikataulutusta kuin kaksi edelle tarkasteltua käyttöjärjestelmää. Kuitenkin käyttäjä voi itse valita käyttääkö se saman prioriteetin omaavien prosessien valintaan kiertovuorottelua vai ei. [8]

Lisäksi käyttäjä voi valita olla käyttämättä prosessien prioriteetteja. Tällöin prosessit eivät koskaan voi tulla ohitetuiksi, prosessi voi toisaalta itse väistyä. [8] Siis prosessien kirjoittamisessa otetaan huomioon mahdolliset ohitukset, mutta käyttöjärjestelmä ei vastaa prioriteettien luomisesta.

Kuten µC/OS-III, freeRTOS ei myöskään tarjoa mahdollisuutta etuoikeuskynnyksen käyttämiseen. Toisaalta prioriteetin käyttämättä jättäminen voidaan toteuttaa muillakin käyttöjärjestelmillä asettamalla kaikkien prosessien prioriteetit samoiksi.

3.4 Analyysi

Ajan hallinta kaikilla kolmella RTOS-järjestelmällä on pääpiirteiltään samankaltainen. Eri järjestelmillä on kuitenkin omia ominaisuuksia, joita toisilla järjestelmillä ei ole. Kootaan eri ominaisuudet taulukkoon 1.

	µC/OS-III	ThreadX	freeRTOS
Prosessien priorisointi	Kyllä	Kyllä	Valinnainen

Etuoikeutettu priorisointi	Kyllä	Kyllä	Valinnainen
Saman prioriteetin käsittely	Kiertovuorottelu aikaviipaleella	Kiertovuorottelu aikaviipaleella/ Jono	Kiertovuorottelu aikaviipaleella tai ilman
Etuoikeuskynnys	Ei	Valinnainen	Ei

Taulukko 1. Vertailtavien reaaliaikaisten käyttöjärjestelmien ajan hallinnan ominaisuuksia.

Ominaisuuksien lisäksi eri järjestelmien väliltä löytyy muitakin eroja. Vertailtavista järjestelmistä µC/OS-III on maksullinen, Eclipse ThreadX ja freeRTOS puolestaan ovat maksuttomia.

Työtä tehdessä vertailuun käytettiin järjestelmien omien dokumentaatioita. Parhain dokumentaatio oli µC/OS-III:n dokumentaatio, vaikka freeRTOS:n dokumentaatio oli lähes yhtä hyvä. Toisaalta ThreadX-järjestelmän dokumentaatio oli melko vaikeasti luettavissa, maksullinen dokumentaatio olisi ollut myös olemassa. Maksullinen versio olisi todennäköisesti ollut mukavampi lukea.

4. YHTEENVETO

Tässä työssä tarkasteltiin kolmen RTOS-järjestelmän aikataulutusta, jotka vastaavat toisiaan. Eri järjestelmien välillä on pieniä eroja, mutta kaikki toimivat suurelta osin samalla tavalla, prioriteettiin pohjautuvalla etuoikeutetulla aikataulutuksella.

Työn tuloksena saatiin selvitettyä, että reaaliaikaisten käyttöjärjestelmien aikataulutukseen käytettään prioriteettiin pohjautuvaa etuoikeutettua aikataulutusta. Tämän avulla käyttöjärjestelmä voi aikatauluttaa prosessit laitteiston laskentaresurssille. Aikataulutus takaa koko järjestelmän toiminnan, sillä kriittiset prosessit saavat halutessaan laskentaresurssit käyttöönsä.

Käyttöjärjestelmät yleisesti myös ratkaisevat mahdollisia ongelmia, kuten lukkiutuminen, jossa kaikki prosessit odottavat toista prosessia, mutta mikään prosessi ei pääse aloittamaan laskentaa. Lisäksi prioriteettien dynaaminen muuttaminen ajon aikana mahdollistaa nälkiintymisen estämisen. Toisaalta dynaaminen prioriteettien muuttaminen vaikeuttaa järjestelmän ennakoitavuutta, joten reaaliaikaisissa järjestelmissä tämä ei ole suositttu ratkaisu.

Työn tutkimuskysymyksiin saatiin vastattua. Käyttöjärjestelmät aikatauluttavat laitteiston laskentaresurssit eri prosessien kesken käyttäen prosessien jakamista prioriteettien mukaan sekä käyttäen kiertovuorottelua. Lisäksi reaaliaikaisten käyttöjärjestelmien ajan hallinta toteutetaan pääosin käyttäen prioriteettiin pohjautuvaa etuoikeutettua aikatalutusta.

Koska aikataulutuksen toteutus on kaikissa kolmessa tutkitussa RTOS-järjestelmässä samanlainen, voidaan eroja etsiä käyttöjärjestelmien muista ominaisuuksista kuten suorituskyvyn tai muistinkäytön mukaan. Kuitenkin tämä vertailu jää tämän työn ulkopuolelle.

LÄHTEET

[1] P. A. Laplante, *Real-time systems design and analysis*, 3rd ed. Hoboken, N.J: Wiley, 2004.

[2] T. Sewell, F. Kam, ja G. Heiser, ”High-assurance timing analysis for a high-assurance real-time operating system”, *Real-Time Syst.*, vsk. 53, nro 5, ss. 812–853, 2017, doi: 10.1007/s11241-017-9286-3.

[3] ”What Are Real-Time Embedded Systems”. Viitattu: 28. maaliskuuta 2024. [Verkossa]. Saatavissa: http://www.cs.uni.edu/~mccormic/RealTime/what.html

[4] C. Garre, D. Mundo, M. Gubitosa, ja A. Toso, ”Real-Time and Real-Fast Performance of General-Purpose and Real-Time Operating Systems in Multithreaded Physical Simulation of Complex Mechanical Systems”, *Math. Probl. Eng.*, vsk. 2014, ss. 1–14, 2014, doi: 10.1155/2014/945850.

[5] I. Haikala, *Käyttöjärjestelmät*. teoksessa Valikko. Helsinki: Talentum, 2003.

[6] ”https://www.analog.com/media/en/dsp-documentation/software-manuals/Micrium-uCOS-III-UsersManual.pdf”. Viitattu: 12. huhtikuuta 2024. [Verkossa]. Saatavissa: https://www.analog.com/media/en/dsp-documentation/software-manuals/Micrium-uCOS-III-UsersManual.pdf

[7] ”rtos-docs/rtos-docs/threadx/threadx-smp/chapter3.md at main · eclipse-threadx/rtos-docs · GitHub”. Viitattu: 29. huhtikuuta 2024. [Verkossa]. Saatavissa: https://github.com/eclipse-threadx/rtos-docs/blob/main/rtos-docs/threadx/threadx-smp/chapter3.md

[8] ”Free RTOS Book and Reference Manual”, FreeRTOS. Viitattu: 10. toukokuuta 2024. [Verkossa]. Saatavissa: https://www.freertos.org/Documentation/RTOS_book.html

[9] W. Ecker, R. Dömer, ja W. Mueller, *Hardware-dependent Software: Principles and Practice*, 1. Aufl. Dordrecht: Springer Netherlands, 2009. doi: 10.1007/978-1-4020-9436-1.

[10] F. Rammig *ym.*, ”Basic Concepts of Real Time Operating Systems”, teoksessa *Hardware-dependent Software*, Dordrecht: Springer Netherlands, 2009, ss. 15–45. doi: 10.1007/978-1-4020-9436-1_2.

[11] ”Context Switch definition”. Viitattu: 5. toukokuuta 2024. [Verkossa]. Saatavissa: https://web.archive.org/web/20100218115342/http://www.linfo.org/context_switch.html

[12] U. Shafi *ym.*, ”A Novel Amended Dynamic Round Robin Scheduling Algorithm for Timeshared Systems”, *Int. Arab J. Inf. Technol.*, vsk. 17, nro 1, ss. 90–98, 2020, doi: 10.34028/iajit/17/1/11.

[13] M. Lab, ”Least Laxity First (LLF) Scheduling Algorithm”, Microcontrollers Lab. Viitattu: 10. huhtikuuta 2024. [Verkossa]. Saatavissa: https://microcontrollerslab.com/least-laxity-first-llf/

[14] R. Mohanty, S. R. Behera, ja S. C. Pradhan, ”A Priority Based Dynamic Round Robin with Deadline (PBDRRD) Scheduling Algorithm for Hard Real Time Operating System”, *Int. J. Adv. Res. Comput. Sci.*, vsk. 3, nro 3, 2012.

[15] Y. Wang ja M. Saksena, ”Scheduling fixed-priority tasks with preemption threshold”, teoksessa *Proceedings Sixth International Conference on Real-Time Computing Systems and Applications. RTCSA’99 (Cat. No.PR00306)*, joulu 1999, ss. 328–335. doi: 10.1109/RTCSA.1999.811269.