ONDERZOEKSVERSLAG RTOS

RTOS

AUTEURS:	MAARTEN WASSENAAR, FRANK TAMER, LEON ZHANG, JEFFREY DE WAAL EN ARSALAN ANWARI
ONDERZOEKSVERSLAG:	THEMAOPDRACHT DEVICES
BEGELEIDER:	DHR ZUURBIER
DATUM:	UTRECHT, 31 OKTOBER 2017

INHOUDSOPGAVE

1 INLEIDING

Voor het thema opdracht Devices maken wij gebruik van Arduino RTOS dat beschikbaar is gesteld door de Hogeschool Utrecht. Indien het gemaakte ontwerp zou moeten werken op een systeem dat niet de Arduino RTOS ondersteunt, moeten we gaan kijken welk ander RTOS daarvoor wel geschikt is. In ons onderzoek gaan we het Arduino RTOS vergelijken met andere RTOS-en. Uiteindelijk gaan we aan de hand van een aantal criteria een keuze maken welk RTOS het beste alternatief is en in hoeverre deze de mechanismen bevat van het Arduino RTOS. In het geval dat een aantal mechanismen niet aanwezig zijn bij het gekozen RTOS, gaan we ook kijken hoe we deze wel kunnen realiseren.

1.1 HOOFDVRAAG

Door te kijken naar welke mechanismen het Arduino RTOS bevat en deze vervolgens te vergelijken met de mechanismen die de andere RTOS-en aanbieden, zijn wij tot de volgende hoofdvraag gekomen:

Met behulp van welke open source real-time operating system kunnen tasks en de concurrency mechanismen, pool, channel, flag(group), clock timer en mutex, zoals aangeboden door het Arduino RTOS, met zo weinig mogelijk overhead worden gerealiseerd?

1.2 DEELVRAGEN

Om de hoofdvraag uiteindelijk te beantwoorden, hebben wij deelvragen opgesteld, zodat wij ieder deel van de hoofdvraag apart aanpakken. Het is belangrijk om te weten wat het Arduino RTOS doet en wat de andere RTOS-en aanbieden. In het geval dat een RTOS een mechanisme mist, gaan we kijken of deze te realiseren is met de mechanismen die het RTOS zelf bevat en in het geval dat het mogelijk is, op welke manier deze dan te realiseren is. Vanuit dit beeld zijn wij tot de volgende deelvragen gekomen:

1. Wat zijn de kenmerkende eigenschappen van tasks en de concurrency mechanismen van het Arduino RTOS?
2. Welke open source RTOS-en zijn beschikbaar?
3. Welk van de beschikbare RTOS-en biedt de meeste van de concurrency mechanismen van het Arduino RTOS aan zonder enige modificatie en biedt dezelfde functionaliteit om taken te realiseren?
4. Welke mechanismen van het Arduino RTOS worden niet ondersteund door de beschikbare RTOS-en?
5. Hoe kunnen de mechanismen van het Arduino RTOS die niet direct worden ondersteund door de beschikbare RTOS-en worden gerealiseerd m.b.v. van deze RTOS-en?

2 ONDERZOEKSMETHODEN

Bij ons onderzoek gaan wij gebruik maken van literatuurstudie als onderzoeksmethode voor de deelvragen 1 t/m 5. Hiervoor maken wij voornamelijk gebruik van de documentatie geleverd door de ontwikkelaars van de verschillende RTOS-en. Wij gaan ieder RTOS afzonderlijk behandelen in de resultaten. Uiteindelijk gaan we de RTOS-en beoordelen op hun updatebeleid, documentatie en concurrency mechanismen. Bij het updatebeleid gaan wij niet alleen kijken naar de frequentie van de updates, maar ook naar andere aspecten, bijvoorbeeld of er toelichting is gegeven bij de updates en of er naar de gebruikers van het RTOS is gecommuniceerd over de update. Voor de hoofdvraag zullen wij de onderzoeksmethode vergelijken bestaande oplossingen toepassen. We gaan kijken welke van deze oplossingen vergelijkbaar zijn met het Arduino RTOS en zullen uiteindelijk een conclusie trekken aan de hand van de resultaten van de deelvragen.

2.1 RTOS CRITERIA

Een RTOS, welke uitgekozen is als vergelijking, moet voldoen aan de volgende eisen:

- Het moet werken op een Cortex M3 platform.
- Het moet open source zijn.
- Het moet C en/of C++ ondersteuning bieden.
- Het moet afgelopen jaar minimaal eenmaal geupdate zijn en actief ontwikkeld worden.
- Het moet over een uitgebreide documentatie beschikken.

3 RESULTATEN

Hier staan de resultaten in verwerkt van elke deelvraag. ‘ Eigenschappen concurrency mechanismen Arduino RTOS’ is hierbij het antwoord op deelvraag 1, waarvoor we de documentatie van Wouter van Ooijen hebben gebruikt. Aan de hand van de zoekmachine Google en de encyclopedie Wikipedia, hebben wij deelvraag 2 beantwoord. Het resultaat hiervan is te vinden onder het kopje ‘Beschikbare open source RTOS’. Onder ‘Mechanismen die ondersteund worden’ hebben wij een tabel opgesteld wat antwoord geeft op deelvraag 3. Om de deelvragen 4 en 5 te beantwoorden zijn wij gaan kijken naar de documentaties van de gekozen RTOS-en en hebben wij de uitwerkingen van beiden deelvragen uiteindelijk geplaatst onder ‘Realisatie van niet ondersteunende mechanismen’.

3.1 EIGENSCHAPPEN CONCURRENCY MECHANISMEN ARDUINO RTOS

Arduino RTOS is het RTOS (wovo/rtos, 2017)[1] dat door Wouter van Ooijen is ontwikkeld voor gebruik in Thema Opdracht Devices en voor gebruik in de C++ programmeerlessen van Wouter. Dit RTOS zal de basis vormen voor de vergelijking met de andere open source RTOS-en. Het RTOS dat hier ter sprake komt is non-preemptive.

3.1.1 TASKS

De gebruiker stelt een lijst van taken op. RTOS werkt aan de hand van de opgegeven taken, iedere taak wacht of op een sein van een andere taak of totdat een bepaalde tijd is verstreken. Dit gebeurt door de *wait()* methode. Wanneer de *wait* is afgelopen, wordt deze gemarkeerd als *ready* en wordt verder gegaan met deze taak. Taken kunnen ook prioriteiten hebben (hoe lager het getal, hoe hoger de prioriteit) en de taken met de hoogste prioriteiten worden zo snel mogelijk uitgevoerd wanneer deze daartoe de beschikking krijgt. Een taak kan pas uitgevoerd worden wanneer deze zich in de status van *resumed* en *non-waiting* bevindt.

3.1.2 WAITABLE

Abstract object waar een taak op kan wachten. Bij RTOS zijn *flag*, *clock*, *timer* en *channel* concrete klassen die overerven van de abstracte klasse *Waitables*. Een *Waitable* is altijd gemaakt voor een bepaalde taak.Verder kan de *Waitable* zich in twee statussen bevinden, *cleared* en *set* (standaard op *clear*).

[1] https://github.com/wovo/rtos

3.1.3 FLAG

Basisch synchronisatie mechanisme. Meerdere taken kunnen hiermee aan een enkele taak doorgeven dat een bepaald event is opgetreden, zonder dat daarbij overdracht plaatsvindt van data. Een *flag* is initieel *clear*. Wanneer een taak wacht op een *flag* en die *flag* wordt *set*, zal de *wait call* de *flag clearen* en zal die taak verder gaan.

3.1.4 (ONE-SHOT) TIMER

De *timer* die hier wordt gebruikt is een speciale type van een *flag.* De *timer* kan namelijk instructies krijgen om na een bepaalde tijd zichzelf op *set* te zetten. Wanneer een timer dat al bezig is met aftellen opnieuw op *set* wordt gezet, wordt de vorige *timeout* overschreven door de nieuwe timer. De timer kan ook onderbroken worden door de *cancel call.*

3.1.5 CLOCK

De *clock* is een mechanisme dat altijd loopt en deze zet zichzelf op *set* na een bepaalde tijd die hij heeft meegekregen.

3.1.6 CHANNEL_BASE

De *channel_base* is een *waitable data queue*. Een *channel_base* is bedoeld voor één taak. Deze taak wacht bij een *channel_base* tot deze *non-empty* is en voert hier een *read* op uit. Alleen een *read* dat resulteert in een lege queue zal de *channel_base* clearen.

3.1.7 MUTEX

Een mutual exclusion semaphore is een synchronisatie mechanisme dat ervoor zorgt dat een taak exclusief toegang verkrijgt tot een bepaalde resource, waarop gegarandeerd wordt dat andere taken geen toegang hebben tot deze resource wanneer deze in gebruik is door een taak. Initieel is een *mutex free.*

3.1.8 MAILBOX_BASE

Een *mailbox* kan gebruikt worden om waarden tussen taken uit te wisselen, deze zijn niet gebonden aan een specifieke taak. Wanneer taak 1 schrijft naar een mailbox, zal deze wachten tot taak 2 deze uitleest. Als taak 2 uit de mailbox leest en niks kan lezen, zal deze wachten tot taak 1 iets in de mailbox schrijft.

3.1.9 POOL_BASE

Een pool is een mechanisme om data uit te wisselen, zonder enige vorm van synchronisatie. Pool kent twee methoden, read() en write(), die garanderen dat ze *atomic* zijn.

3.2 BESCHIKBARE OPEN SOURCE RTOS

Wij hebben de zoekmachine Google gebruikt om een lijsten (Open Source RTOS, 2017; Comparison of real-time operating systems, 2017; RTOS LIST – EmbeddedCraft. 2016)[2] te vinden van beschikbare RTOS-en. Vervolgens zijn we een flink aantal RTOS-en langsgegaan en hebben we gekeken welke RTOS-en voldoen aan de criteria die wij in onze 'Methode' hebben beschreven. Uiteindelijk hebben we een kleine selectie van 6 gemaakt waarvan wij denken dat deze relevant zijn voor het onderzoek. We gaan in dit hoofdstuk voor elk gekozen RTOS algemene informatie uitwerken.

3.2.1 CHIBIOS (2006)[3]

ChibiOS/RT is een compact en snel RTOS, ontwikkeld door Giovanni Di Sirio en uitgegeven onder de GPL3 licentie. Dit RTOS is ontworpen om te werken met embedded systemen op basis van een 8,16 en 32-bits microcontroller en is ook geschikt voor het Cortex-M3 platform. Met ChibiOS kan in C en Assembly geprogrammeerd worden, wat veel mogelijkheden biedt. ChibiOS ondersteunt *preemptive multithreading scheduling* en R*ound-Robin scheduling voor threads met hetzelfde prioriteit*.

ChibiOS bevat een uitgebreide documentatie die alle functies goed en duidelijk beschrijft en wordt door middel van GitHub (https://github.com/ChibiOS, 2017) goed up-to-date gehouden. Naast GitHub heeft ChibiOS ook een eigen website (http://www.chibios.org/dokuwiki/doku.php, 2006) met aanvullende informatie over dit gehele RTOS. Verder heeft ChibiOS een goede band met de community. Zo kan iedereen bijdragen aan de ontwikkeling van dit RTOS. Na de contributie blijven alle rechten voor de geschreven code voor de community.

3.2.2 DISTORTOS (2014)[4]

DistortOS is een object-oriented C++ RTOS voor microcontrollers die geschreven is in C++11. DistortOS is momenteel geschikt voor alle ARMv6-M en ARMv7-M architecturen, wat betekent dat dit RTOS erg veelzijdig is. Ook is dit RTOS geheel *preemptive* en ondersteunt zowel *Round-Robin scheduling*, als ook *FIFO scheduling*. Het RTOS probeert de functionaliteiten zo efficiënt en universeel mogelijk te implementeren. Dit RTOS probeert daarom ook niet de snelste of kleinste RTOS te zijn, waardoor er eventuele overhead kan ontstaan. DistortOS richt zich ook op het implementeren van de beste functionaliteiten van andere open source RTOS-en.

De documentatie is op GitHub (https://github.com/DISTORTEC/distortos, 2014) te vinden, naast GitHub is er een veel uitgebreidere documentatie beschikbaar op de website van DistortOS met allerlei foto's en video's over hoe je de functionaliteiten van dit RTOS kunt gebruiken.

Bij DistortOS is er veel interactie tussen de community en de developers. Je kan de developers mailen voor informatie en eventueel maandelijks een nieuwsbrief ontvangen. Na elke grote update doen de developers een vragensessie.

[2] https://www.osrtos.com/ , https://en.wikipedia.org/wiki/Comparison_of_real-time_operating_systems , http://www.embeddedcraft.org/listrtos.html
[3] https://en.wikipedia.org/wiki/ChibiOS/RT
[4] http://distortos.org/

3.2.3 ECOS (2009)[5]

eCos is een opensource RTOS dat gebruikt wordt voor applicaties die heel diep in het embedded systeem zitten. Dit RTOS wordt gebruikt door grote internationale bedrijven zoals Sony, NETGEAR. Dit RTOS gebruikt een HAL gebaseerd op C en heeft een C/C++ toolset. Het ondersteunt bijna alle 32-bits architecturen, wat er voor zorgt dat de flexibiliteit heel hoog ligt. Deze flexibiliteit is echter niet toepasselijk voor ARM based architecturen.

Het update beleid van dit RTOS is matig, omdat het eens in de zoveel maanden een andere platform update of een nieuwe versie uitkomt. Verder is de documentatie (http://ecos.sourceware.org/ecos/docs.html, 2001) van deze RTOS onvoldoende, omdat er geen handleiding aanwezig is waar de functies goed uitgelegd zijn. Er is een enkele pagina die verteld dat de mechanismen bestaan.

3.2.4 MBED-RTOS (2017)[6]

Mbed-RTOS is een RTOS gebaseerd op de RTX-implementatie van de CMSIS-RTOS API. Dit RTOS is geschreven in C++ en kan gebruikt worden door alle Cortex-M varianten. Ook kun je SVD(XML) files gebruiken van chipmakers en deze gebruiken als een header-file. Dit is de meest uitbreidbare API in dit verslag, omdat het door verschillende RTOS-vendors ondersteund wordt. Dit zorgt er echter wel voor dat het niet het meest efficiënte RTOS is. Wel is het het meeste flexibele RTOS, omdat je gegevens van meerdere chipmakers kan verwerken via de API.

Mbed-RTOS maakt gebruik van priority based scheduling, wat betekent dat een taak met de hoogste prioriteit eerst door de processor wordt uitgevoerd.

Het update beleid van dit RTOS is zeer goed, ARM-Mbed heeft een actieve community met een Q&A forum (https://os.mbed.com/forum/mbed/, 2017) om problemen en vragen te beantwoorden. Verder bezit ARM-Mbed over meer Hardware API's en libraries dan alleen Mbed-RTOS.

3.2.5 NUTTX (2007)[7]

Nuttx is een open source RTOS geschreven in C, C++ en assembler. Dit RTOS is gemaakt door Gregory Nutt in 2007 en uitgegeven onder de BSD licentie. Dit RTOS is geschikt voor microcontrollers van 8 tot 32-bits. Het RTOS is geheel preemptive en ondersteund de meeste ARM architecturen, wat zorgt voor veel flexibiliteit voor de gebruikers. Daarnaast bevat NuttX ook een FIFO-, Round-Robin- en sporadic scheduling. Bij dit RTOS ligt configuratie hoog in het vaandel, waardoor de gebruiker veel zelf kan configureren.

De documentatie (http://nuttx.org/doku.php?id=documentation, 2007) van NuttX is volledig. Het bevat informatie over wat elke methode doet en geeft daarbij ook aan welke parameters deze mee krijgt en wat de eventuele returnwaardes zijn.

Interactie tussen de developer en de community is er in de vorm van een Yahoo forum (https://groups.yahoo.com/neo/groups/nuttx/info, 2008). Dit is wel de enige plek voor de community om informatie te delen of vragen te stellen.

[5] http://ecos.sourceware.org/
[6] https://os.mbed.com/handbook/RTOS
[7] http://nuttx.org/doku.php?id=nuttx

3.2.6 STATEOS (2015)[8]

StateOS is een simpel en relatief snelle RTOS. De documentatie, te vinden op hun GitHub, is matig, maar wordt wel ondersteund met voorbeelden. Verder wordt StateOS nog regelmatig bijgewerkt, maar is er niet echt een forum waar vragen gesteld kunnen worden. Het concurrency mechanisme bevat alles wat verwacht wordt volgens onze criteria. Het is gebouwd uit het concept van de state machine. Verder is dit RTOS geschreven in C++.

3.3 MECHANISMEN DIE ONDERSTEUND WORDEN

Aan de hand van deelvraag 2 hebben wij de gekozen RTOS in een tabel uitgezet om een overzicht te krijgen van de verschillende RTOS-en. Uit dit tabel is te zien dat ChibiOS, DistortOS en StateOS RTOS-en zijn die alle concurrency mechanismen van het Arduino RTOS aanbiedt zonder enige modificatie. Als wij ook kijken naar welke RTOS dezelfde functionaliteiten aanbiedt als het Arduino RTOS om taken te kunnen realiseren dan is Mbed-RTOS ook zeer geschikt.

Naam	flag	timer	clock	mutex	channel	mailbox	pool
Arduino RTOS	✓	✓	✓	✓	✓	✓	✓
ChibiOS/RT	✓	✓	✓	✓	✓	✓	✓
DistortOS	✓	✓	✗	✓	✓	✗	✗
eCos	✓	✓	✓	✓	✗	✓	✓
Mbed-RTOS	✓	✓	✗	✓	✓	✓	✓
NuttX	✓	✓	✗	✓	✓	✗	✗
StateOS	✓	✓	✓	✓	✓	✓	✓

Tabel 1. Mechanismen van de gekozen RTOS-en

3.4 REALISATIE VAN NIET ONDERSTEUNDE MECHANISMEN

In de tabel van deelvraag 3 kunnen we direct zien welke mechanismen ondersteund worden en welke niet. Wij zullen per RTOS antwoord geven op de deelvraag welke mechanismen niet ondersteund worden en hoe dat eventueel gerealiseerd zou kunnen worden.

3.4.1 CHIBIOS

Binnen ChibiOS worden alle concurrency mechanismen ondersteunt die ook door Arduino RTOS worden ondersteunt. Er worden andere namen gebruikt voor een aantal mechanismen. Voor *clock* wordt de methode *Tickless Mode* gebruikt onder de methode *Virtual Timers*, *Messages* voor *channel* en *Memory Pool* voor *pool*. Hierdoor kunnen wij zonder mechanismen om te bouwen dit RTOS gebruiken.

3.4.2 DISTORTOS

DistortOS ondersteunt de *clock*, *mailbox* en *pool* niet. De *clock* is makkelijk op te lossen door een *timer* oneindig door te laten lopen, zoals de *clock* dat ook doet in het Arduino RTOS. *pool* is op te lossen door zelf een struct (die een single value bevat) aan te maken waar naar geschreven en uit gelezen kan worden. Hetzelfde geldt ook voor mailbox, alleen is het net wat ingewikkelder, omdat ze synchroon lopen en omdat read() en write() op elkaar moeten wachten.

[8] https://github.com/stateos/StateOS

3.4.3 ECOS

Om de stabiliteit van dit RTOS te garanderen, worden niet alle concurrency mechanismen ondersteund. Een *channel* is een heel intensieve mechanisme die moeilijk te implementeren is voor kleine microcontrollers. In principe kan je dit probleem oplossen door een array te gebruiken.

3.4.4 MBED-RTOS

Het enige nadeel van dit RTOS is dat het geen *clock* mechanisme bevat. In principe kan je dit heel simpel zelf realiseren door een *timer* een maal te setten en vervolgens constant in een loop te zetten. Ook is het mogelijk om een methode toe te voegen aan de thread / taak die ervoor zorgt dat hij oneindig wacht.

3.4.5 NUTTX

NuttX ondersteunt net als DistortOS geen *clock*, *pool* en *mailbox*. Ook hier geldt dat *clock* makkelijk te implementeren is door de parameter *it_interval* een extreem hoge waarde te geven in de methode *timer_settime* van NuttX. De mechanismen *pool* en *mailbox* zijn op dezelfde manier te implementeren als beschreven in paragraaf 3.4.2 DistortOS.

4 CONCLUSIE

Wij hebben al onze geselecteerde RTOS-en in een tabel hieronder verwerkt en scores gegeven op basis van het update-beleid, de documentatie en of de RTOS-en de nodige concurrency mechanismen bevatten om de taken van het Arduino RTOS te kunnen realiseren.

RTOS	UPDATEBELEID	DOCUMENTATIE	CONCURRENCY MECHANISMEN	COMMUNITY
ChibiOS/RT	G	G	G	V
DistortOS	V	G	O/V	V
eCos	O	O	V	O
Mbed-RTOS	G	G	V/G	G
NuttX	V	G	O/V	V
StateOS	G	O	G	V

Tabel 2. Scoretabel (O = onvoldoende, V = voldoende, G = goed)

Uit de scores is te zien dat er drie RTOS'en zijn die goed scoren voor het updatebeleid. Dit houdt in dat de ontwikkelaars voor deze RTOS-en actief bezig zijn met het verbeteren van hun RTOS en dat gebruikers eventueel feedback kunnen geven (of bugs kunnen reporten) die de ontwikkelaars kunnen meenemen naar de volgende update.

De documentatie zijn voor de meeste RTOS-en wel goed uitgewerkt. Er staat goed beschreven hoe klassen in elkaar zaten en welke methoden ze bevatten en wat deze methoden doen. Zowel eCos als ook StateOS hebben hier onvoldoende gescoord, omdat ze vergeleken met de andere RTOS-en weinig of geen documentatie geven.

Voor de concurrency mechanismen scoort de helft van onze RTOS-en een voldoende of lager. De overhead leek bij ons te groot bij deze RTOS-en om de mechanismen te vervangen van Arduino RTOS. Bij Mbed-rtos mist er maar 1 mechanisme, die je gemakkelijk zelf kan implementeren, vandaar dat we deze hebben geplaatst onder de grens van voldoende en goed. Bij twee van de RTOS-en zijn alle concurrency mechanismen aanwezig, waarbij sommige andere namen voor hun mechanismen gebruiken.

Het laatste aspect waar wij naar hebben gekeken is de community. Hier scoren de meeste RTOS-en wel een voldoende op, op eCos na en Mbed-RTOS na. eCos heeft op dit moment alleen een mailsysteem en dit is vergeleken met de rest van de RTOS-en wel een enorme minpunt, omdat je geen zekerheid hebt of je mail behandeld wordt of niet. Mbed-RTOS daarentegen scoort als enige goed in tegenstelling tot de rest, omdat ze naast een Q&A forum ook nog fora bevatten in verschillende talen.

Wij concluderen uit dit tabel dat Mbed-RTOS het meest geschikt is om met zo weinig mogelijk overhead de tasks en concurrency mechanismen, zoals aangeboden door het Arduino RTOS, te realiseren. Zoals eerder vermeld zorgt een goede updatebeleid dat dit RTOS steeds verbeterd wordt. Verder is er een goede documentatie met codevoorbeelden en plaatjes om duidelijk te weergeven wat hun mechanismen doen en zijn de missende mechanismen relatief makkelijk te realiseren. Dat de community fora meerdere talen ondersteund, maakt het gemakkelijker voor de gebruiker om ergens over mee te praten wanneer dit zijn/haar moedertaal is, al is engels natuurlijk altijd de voorkeur.

5 AANBEVELING

Naast Mbed-RTOS scoorde ChibiOS ook zeer goed. Dit RTOS heeft alle concurrency mechanismen en is daarnaast net als Mbed-rRTOS veelzijdig, bevat goede documentatie en is ook goed actief op het gebied van updates en community.

StateOS leek ons in het begin ook een goede aanbeveling, maar omdat deze te weinig documentatie bevat, raden wij toch een andere RTOS aan. Was de documentatie iets duidelijker en uitgebreider, dan had dit RTOS zeker veel potentie om aanbevolen te worden in dit onderzoek.

6 LITERATUURLIJST

ChibiOS/RT - Wikipedia. (2017). Opgehaald van https://en.wikipedia.org/wiki/ChibiOS/RT

Comparison of real-time operating systems. (2017). Opgehaald van https://en.wikipedia.org/wiki/Comparison_of_real-time_operating_systems

DISTORTEC. (2017). Opgehaald van https://github.com/DISTORTEC

distortos - object-oriented C++ RTOS for microcontrollers. (2014). Opgehaald van http://distortos.org/

eCos. (2009). Opgehaald van http://ecos.sourceware.org/

eCosCentric - the eCos real-time operating system experts. (2002). Opgehaald van http://www.ecoscentric.com/index.shtml

fetch.php. (2017). Opgehaald van http://www.chibios.org/dokuwiki/lib/exe/fetch.php?media=documents:manuals:17.6:chibios_17.6.3_chibios_rt_rm.pdf

Homepage - Handbook | Mbed. (2017). Opgehaald van https://os.mbed.com/handbook/Homepage

NuttX Real-Time Operating System - NuttX Real-Time Operating System. (2007). Opgehaald van http://nuttx.org/

Ooijen, W. v. (2017). *wovo/rtos*. Opgehaald van https://github.com/wovo/rtos

Open Source RTOS. (2017). Opgehaald van https://www.osrtos.com/

RTOS LIST - EmbeddedCraft. (2016). Opgehaald van http://www.embeddedcraft.org/listrtos.html

stateos/StateOS. (2015). Opgehaald van https://github.com/stateos/StateOS