Onderzoeksverslag RTOS

Opdrachtgever: Hogeschool Utrecht
Begeleider: Joop Kaldeway
Klas: V2A
Team: MajorLazor + 2
Opgesteld door: Niels de Waal (1698041), Jasper Smienk (1700502),
Wouter Dijkstra (1700101), Eelke Feitz (1701685),
Nick Bout (1709217)
Datum: 25-10-2017
Versie: 3.0

Voorwoord

Voor de aanvang van dit verslag zouden wij, de leden Majorlasor-2, zowel Jan Zuurbier als Joop Kaldeway willen bedanken voor hun ondersteuning, feedback en begeleiding tijdens dit project.

In dit verslag doen we onderzoek naar het arduino RTOS geschreven door Wouter van Ooijen en andere beschikbare RTOS-en op het internet. Om enige verwarring tussen beide te voorkomen wordt in dit verslag het RTOS van Wouter van Ooijen vermeld als ‘huRTOS’.

Inhoudsopgave

Samenvatting

Wij deden dit onderzoek en schreven dit rapport uit opdracht van de Hogeschool Utrecht voor de themaopdracht devices. In deze opdracht worden we met het probleem geconfronteerd dat het overschrijven van een softwaresysteem, dat gemaakt is voor één RTOS, overhead levert als we het willen overschrijven naar een ander RTOS. De vraagstelling waarmee wij te werk zijn gegaan luidt als volgt: ***(bron 1)***

"Met behulp van welk opensource RTOS kunnen tasks & concurrency mechanismen (pool, channel, flag (group), clock timer & mutex) zoals aangeboden door het huRTOS, met zo weinig mogelijk overhead worden gerealiseerd?"

Voor het beantwoorden van deze hoofdvraag hebben wij in ons rapport gebruik gemaakt van de volgende 5 deelvragen.

1. *Wat zijn de kenmerkende eigenschappen van tasks en de concurrency mechanismen van het Arduino RTOS?*
2. *Welke open source RTOS-en zijn beschikbaar?*
   eisen:
   a. *Het RTOS moet beschikbaar zijn voor het ARM platform.*
   b. *Het RTOS moet C en C++ ondersteunen.*
   c. *Het RTOS moet binnen het afgelopen jaar een update hebben gekregen.*
   d. *Het RTOS moet beschikken over documentatie waarin de geboden functionaliteit is beschreven.*
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?*

Uit het onderzoek is gebleken dat 'FreeRTOS' het RTOS zou zijn waarbij we het minste overhead zouden creëren als we ons systeem hiernaar zouden overschrijven. Daarom raden wij iedereen die gebruikgemaakt heeft van het huRTOS en zijn programma wilt overschrijven naar een ander RTOS hiervoor FreeRTOS te gebruiken.

Deelvraag 1

“Wat zijn de kenmerkende eigenschappen van tasks en de concurrency mechanismen van het Arduino RTOS?”

Inleiding

Deze deelvraag geeft ons informatie en inzicht over de functies die we hebben in het huRTOS. Hierdoor weten we betere eisen te stellen aan het RTOS waar we ons systeem naar willen overschrijven.

Voor het beantwoorden van deze deelvraag hebben we gebruik gemaakt van de onderzoeksmethode “Bieb”. Deze vorm van onderzoek paste het best in het gegeven tijdsbestek, aangezien deze werkvorm snel en efficiënt resultaten levert. Tevens sluit deze werkvorm aan bij de opdracht die we moeten verrichten, namelijk het bestuderen van al bestaande informatie.

Kern / Conclusie

Na het doorlezen van de huRTOS reader en de github source code blijkt dat het huRTOS beschikt over de volgende tasks & concurrency mechanismen. ***(bron 2)***

RTOS Name	Flag	Timer	Clock	Mutex	Channel	Pool	Mailbox	Grouped waitables	task switching
huRTOS	✔	✔	✔	✔	✔	✔	✔	✔	Preemptive

Verder switched het huRTOS non-preemptive tussen zijn taken. Dit betekend dat er alleen tussen taken wordt geswitched wanneer er een RTOS call wordt gemaakt. Dit kan o.a. gedaan worden door “hwlib::wait_(ms)” aan te roepen. Hierdoor worden tasks niet onderbroken tijdens het uitvoeren van hun statements. Wat er voor zorgt dat de data die uitgelezen wordt altijd up to date is.

Om een beeld te krijgen wat deze mechanismen inhouden volgt voor ieder mechanisme een korte uitleg. ***(bron 3)***
(https://sites.google.com/site/rtosmifmim/home/message-queues-mailboxes-and-pipes)
https://sites.google.com/site/rtosmifmim/home/message-queues-mailboxes-and-pipes

Clock
Een clock is een timing element dat een puls afvuurt na een vaste tijdseenheid. Deze vaste tijdseenheid is gedefinieerd bij het aanmaken van de clock. Als deze eenmaal aangemaakt is zal deze door blijven lopen en pulsen afvuren na ieder van deze tijdseenheden tot het programma eindigt.

Timer
Een timer is net als een clock een timing element. Ook deze vuurt na een tijdseenheid een puls af, maar is niet precies hetzelfde. Waar een clock oneindig door blijft lopen kan een

timer stilgezet worden en opnieuw aangezet. Tijdens het aanzetten kan ook de tijdseenheid tussen de pulsen opnieuw ingesteld worden.

Flag
Een flag is een synchronisatie element wat deel is van een grotere task (task A). Deze flag is dus deel van deze task A, echter wordt hier niet door uitgevoerd. Een andere task (task B) kan de flag setten. Task A kan wachten op de flag, wanneer deze geset wordt door task B kan task A weer verder met zijn taak en wordt de flag weer gereset.

Pool
Een pool is een class template die gebruikt wordt als variabele dat gedeeld door verschillende tasks. Het doel hiervan is om voor de gebruiker deze deling tussen de verschillende tasks duidelijk te maken. Hiernaast wordt het makkelijker om de code om te zetten naar een ander platform waar er meer acties nodig zijn om data te delen tussen verschillende tasks.

Mutex
Een mutex, ofwel Mutual Exclusion Flag, is een flag die geclaimd kan worden door een task wanneer deze bij gedeelde data moet zijn. Wanneer de mutex over een deel data geclaimd is kunnen andere tasks niet bij deze data en wachten tot de mutex wordt vrijgegeven. Dit voorkomt dat er door meerdere tasks tegelijkertijd een gedeeld stuk data wordt gelezen en aangepast.

Channel
Een channel is een class template die wordt gebruikt als een array met data elementen. De task die eigenaar is van de channel kan hem uitlezen vanaf de voorkant. Andere tasks kunnen data schrijven naar het einde van de channel. De data die in de channel staat wordt dus uitgelezen via het FIFO systeem, First In First Out. Wanneer er data wordt uitgelezen wordt dit element ook verwijderd uit de channel, waardoor alles ook een positie hierin opschuift naar de voorkant. Wanneer er niets in de channel staat en de task toch deze probeert uit te lezen blokkeert deze task totdat er data aanwezig is. Wanneer een ander task data aan een volle channel probeert toe te voegen wordt dit genegeerd en gaat de data verloren.

Mailbox
Een mailbox is een class template en een synchronisatie mechanisme dat in principe werkt als een doorgeefluik. Een task schrijft data in deze mailbox en wacht totdat een andere taak, waarvoor deze data bedoeld is, deze data uitleest. Net zoals bij een channel wordt de data in de mailbox vernietigd. Als de mailbox al data bevat worden andere tasks geblokkeerd om hier naar te kunnen schrijven.

Grouped waitables
Waitables zijn abstracte objecten, dus bestaan niet op zichzelf. In het huRTOS erven flag, timer, clock en channel van waitable. Deze mechanismen zijn dus waitable en kunnen door tasks op gewacht worden. De tasks in het huRTOS kunnen op één of meerdere, tot maximaal 31 waitables wachten.

Deelvraag 2

“Welke open source RTOS-en zijn er beschikbaar?”

Inleiding

Met deze deelvraag hopen we een beter beeld te krijgen van de verschillende beschikbare RTOS systemen en ook over de kwaliteiten, de mogelijkheden en de functies die deze bieden.

Voor het beantwoorden van deze deelvraag hebben we gebruik gemaakt van de onderzoeksmethode “Bieb”. Deze vorm van onderzoek paste het best in het gegeven tijdsbestek, aangezien deze werkvorm snel en efficiënt resultaten levert. Tevens sluit deze werkvorm aan bij de opdracht die we moeten verrichten, namelijk het bestuderen van al bestaande informatie.

Kern

Het RTOS moet voldoen aan de volgende eisen:

- Het RTOS moet beschikbaar zijn voor het ARM platform.
- Het RTOS moet C en C++ ondersteunen.
- Het RTOS moet binnen het afgelopen jaar een update hebben gekregen.
- Het RTOS moet beschikken over documentatie waarin de geboden functionaliteit is beschreven.

Hieronder een lijst van alle RTOS-en die mogelijk als vervanging kunnen dienen voor het huRTOS.

Name	Open Source	Links
BeRTOS ***(bron 4)***	YES	https://github.com/develersrl/bertos
uKOS	YES	(http://www.ukos.ch/) http://www.ukos.ch/
BRTOS ***(bron 5)***	YES	https://github.com/brtos/brtos
distortos	YES	(http://distortos.org/documentation/) http://distortos.org/documentation/
Dnx RTOS	YES	(http://www.dnx-rtos.org/) http://www.dnx-rtos.org/
eChronos	YES	(https://ts.data61.csiro.au/projects/TS/echronos/) https://ts.data61.csiro.au/projects/TS/echronos/

F9 microkernel ***(bron 6)***	YES	https://github.com/f9micro/f9-kernel
FreeRTOS	YES	(http://www.freertos.org/) http://www.freertos.org/
silRTOS	YES	(http://spanidea.com/sil.html)http://spanidea.com/sil.html
stratifyOS ***(bron 7)***	YES	https://github.com/StratifyLabs/StratifyOS (https://stratifylabs.co/Stratify-OS/) https://stratifylabs.co/Stratify-OS/

Conclusie

Voor ons onderzoek zijn veel verschillende RTOS-en die voldoen aan onze gestelde eisen. Omdat het tijdsbestek waarin wij moeten werken nogal krap is hebben wij ons onderzoek beperkt tot de bovenstaande tien real time operating systems.

Deelvraag 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?"

Inleiding

Voor het beantwoorden van deze deelvraag hebben we gebruik gemaakt van de onderzoeksmethode "Bieb". Deze vorm van onderzoek paste het best in het gegeven tijdsbestek, aangezien deze werkvorm snel en efficiënt resultaten levert. Tevens sluit deze werkvorm aan bij de opdracht die we moeten verrichten, namelijk het bestuderen van al bestaande informatie.

Kern

Om een beeld te krijgen welke RTOS-en dezelfde features bieden als het huRTOS, hebben we een tabel gemaakt waarin staat welke features deze real time operating systems bieden.

RTOS Name	Flag	Timer	Clock	Mutex	Channel	Pool	Mailbox	Grouped waitables	task switching
BeRTOS		✔		✔	✔	✔	✔		Preemptive
uKOS		✔		✔	✔	✔	✔		Preemptive
BRTOS		✔		✔	✔		✔		Preemptive
distortos	✔	✔	✔	✔	✔				Preemptive
Dnx RTOS	✔			✔	✔				Preemptive
eChronos		✔		✔	✔				Preemptive & Cooperative
F9 microkernel		✔			✔				Cooperative
FreeRTOS	✔	✔		✔	✔		✔	✔	Preemptive & Cooperative
stratifyOS		✔		✔	✔				Preemptive

Conclusie

Voor het bepalen welk real time operating system de meeste concurrency mechanismen zonder enige modificatie biedt is het een kwestie van vergelijken en afstrepen. Zo komen we tot de conclusie dat FreeRTOS de meeste concurrency mechanismen van het Arduino RTSO aan te bieden zonder enige modificatie.

Deelvraag 4

“Welke mechanismen van het huRTOS worden niet ondersteund door de beschikbare RTOS-en?”

Inleiding

Voor het beantwoorden van deze deelvraag hebben we gebruik gemaakt van de onderzoeksmethode “Bieb”. Deze vorm van onderzoek paste het best in het gegeven tijdsbestek, aangezien deze werkvorm snel en efficiënt resultaten levert. Tevens sluit deze werkvorm aan bij de opdracht die we moeten verrichten, namelijk het bestuderen van al bestaande informatie.

Kern

Voor het beantwoorden van deze deelvraag bouwen we voort op de conclusie van de vorige. Hier concludeerde we dat FreeRTOS de meest geschikte optie zou zijn om de concurrency mechanisme zoals aangeboden door het huRTOS me zo weinig mogelijk overhead te realiseren.

Conclusie

Ook al is FreeRTOS het meest geschikte RTOS uit ons onderzoek gebleken, mist het toch nog een aantal concurrency mechanismen die het huRTOS wel heeft. Deze mechanismen zijn de clock en de pool.

Deelvraag 5

“Hoe kunnen de mechanismen van het huRTOS die niet direct worden ondersteund door de beschikbare RTOS-en worden gerealiseerd m.b.v. van deze RTOS-en?”

Inleiding

Voor het beantwoorden van deze deelvraag hebben we gebruik gemaakt van de onderzoeksmethode “Bieb”. Deze vorm van onderzoek paste het best in het gegeven tijdsbestek, aangezien deze werkvorm snel en efficiënt resultaten levert. Tevens sluit deze werkvorm aan bij de opdracht die we moeten verrichten, namelijk het bestuderen van al bestaande informatie.

Kern

Pool *(bron 8)*
Een pool zou je kunnen maken door gebruik te maken van een globale variabelen.

Clock *(bron 9 & 10)*
Een clock zou je kunnen maken door gebruik te maken van het bestaande System Tick dat beschikbaar is op ARM processoren.

Conclusie

Het blijkt dus dat we de concurrency mechanismen die het RTOS niet direct ondersteunt kunnen nabootsen met al bestaande mechanismen.

Conclusie

Met behulp van FreeRTOS zouden we de tasks & concurrency mechanismen aangeboden door het Arduino RTOS, met zo weinig mogelijk overhead kunnen realiseren. Dit komt doordat FreeRTOS van alle onderzochte RTOS-en de meest overeenkomende concurrency mechanismen heeft. Hierdoor komt er het minst overhead bij het improviseren van de missende concurrency mechanismen.

Aanbeveling

Aan ieder persoon die een systeem heeft gemaakt op het huRTOS en deze met zo weinig mogelijk overhead zou willen overschrijven naar een ander RTOS zou ik het gebruik van FreeRTOS hiervoor aanraden. Dit RTOS heeft de meest overeenkomende concurrency mechanismen, wat het overschrijven het makkelijkst maakt om binnen een kort tijdsbestek.

Bronnenlijst

1. Zuurbier, J. (2016). *V2THDE - Casus Lasertag 2016 - 2017*. Geraadpleegd van https://cursussen.sharepoint.hu.nl/fnt/36/TCTI-V2THDE-16/Studiemateriaal/Casus/V2THDE%20-%20Casus%20lasertag%202016-2017.pdfv
2. Ooijen, W. v. (2017, 12 juli). RTOS [GitHub repository]. Geraadpleegd op 30 oktober 2017, van https://github.com/wovo
3. Ooijen, W. v. (2017). *2018-2018-V2CPSE1-reader-rts*. Geraadpleegd van https://cursussen.sharepoint.hu.nl/fnt/35/TCTI-V2CPSE1-16/default.aspx
4. Basile, D., Sacchi, F., Fedrigo, S., Ottaviano, L., Italia, M., Rainone, A., . . . Teisberg, T. (2004, 23 mei). beRTOS [GitHub repository]. Geraadpleegd op 30 oktober 2017, van https://github.com/develersrl/bertos
5. D, W.-G., & Barriquello, C.-H. (2014, 8 december). bRTOS [GitHub repository]. Geraadpleegd op 30 oktober 2017, van https://github.com/brtos/brtos
6. Huang, J., Hung, S., Hsiao, V., Liaw, J.-F., Liao, W., Ammar, A., . . . Som, L. (2013, 15 juli). F9-kernel [GitHub repository]. Geraadpleegd op 30 oktober 2017, van https://github.com/f9micro/f9-kernel
7. Gilbert, T. (2016, 2 mei). StratifyOS [GitHub repository]. Geraadpleegd op 30 oktober 2017, van https://github.com/StratifyLabs/StratifyOS
8. Schalken-Pinkster, J., & Zuurbier, J. (2017, 6 oktober). V2CSM-16-week-5-B [PowerPoint presentatie]. Geraadpleegd van https://cursussen.sharepoint.hu.nl/fnt/35/TCTI-V2CSM1-16/default.aspx
9. Norris, S. (2014, 27 juli). Trying to Create Clock at Runtine [Blogpost]. Geraadpleegd van https://e2e.ti.com/support/embedded/tirtos/f/355/t/351311
10. Joe, B. (2016, 18 mei). The FreeRTOS Tick – Unravel the Mystery to Master the Tick [Blogpost]. Geraadpleegd van http://www.learnitmakeit.com/freertos-tick/