Hogeschool Utrecht

Themaopdracht Devices

2018_TCTI-V2THDE-16_1_V, Groep 1

Onderzoeksrapport
Real-Time Operating Systems

Auteurs:
Cris van der Nolle
Nathan Hoekstra
Nick Snel
Ramon van Bemmel
Marc Dirven

Docent:
Jan Zuurbier

Inhoudsopgave

1 Inleiding

1.1 Aanleiding onderzoek

In de cursus Themaopdracht Devices, dient er een Real Time Operating System gebruikt te worden. De reden van dit onderzoek is om meer kennis op te doen over verschillende Real Time Operating Systems. Dit onderzoek zal verschillende RTOS-en bespreken.

1.2 RTOS

Om te kunnen begrijpen wat de onderzoeksvraag precies inhoudt, dienen wij eerst de betekenis van Real-Time Operating System (in ICT jargon ook wel RTOS genoemd) goed te formuleren.

"Een Realtimebesturingssysteem (Eng: Real-time Operating System, RTOS) is een besturingssysteem waarin de realtime-aspecten de nadruk hebben. Dit wil zeggen dat taken door het besturingssysteem uitgevoerd kunnen worden op door de gebruiker aangeduide tijdstippen en met een door de gebruiker opgegeven prioriteit. Realtime-besturingssystemen worden voornamelijk in toepassingsgerichte omgevingen gebruikt"[1]

1.3 Arduino RTOS

Het Arduino RTOS is een RTOS dat wordt aangeboden door de Hogeschool Utrecht, geschreven door Wouter van Ooijen. Het wordt dus ook wel HU RTOS genoemd. Het RTOS bevat onder andere pools, flags, timers, clocks, channels en meer. Het is in object georiënteerd C++ geschreven en is bedoeld voor ARM processoren, waaronder dus de Arduino.

1.4 Hoofdvraag onderzoek

De onderzoeksvraag luidt:

"Met behulp van welk Open Source Realtime 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?"

De hoofdvraag dient te worden opgesplitst in deelvragen. Er worden verschillende eisen aan de deelvragen gesteld, zoals:

- geen deelvragen die niet nodig zijn voor het beantwoorden van de hoofdvraag;
- deelvragen dienen minder complex te zijn dan de hoofdvraag;
- deelvragen dienen in een logische volgorde te staan;
- deelvragen dienen verhalend of op een complexiteit volgorde te staan, zodat ze een lopend verhaal creëren

1.4.1 Deelvragen

De deelvragen luiden als volgt (klik op een sectie om naar de deelvraag te gaan):

- Wat zijn de kenmerkende eigenschappen van tasks en de concurrency mechanismen van het Arduino RTOS?
- Welke open source RTOS-en zijn beschikbaar?
- 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?
- Welke mechanismen van het Arduino RTOS worden niet ondersteund door de beschikbare RTOS-en?
- 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?

Bij al deze deelvragen passen wij de onderzoeksmethode, bieb toe. Bieb omvat onder andere het vergelijken van al bestaande oplossingen, en het raadplegen van literatuurstudie.

2 Deelvraag 1

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

2.1 Toelichting

Een ding dat ons opviel, was dat het HU-RTOS een aantal Assembly files bevat. Als we gaan kijken naar het FreeRTOS, waar we later nog op terugkomen, wordt er geen Assembly gebruikt. Ook valt ons op dat er moet worden overgeërfd van `rtos::task<>`. Bij sommige RTOS-en kun je gewoon een object aanmaken, en programmeren wat deze routine moet gaan doen.

Ook wordt er bij het HU-RTOS het datatype "Pool" gebruikt. Hiermee kun je slechts één variabele van het abstracte datatype `<typename T>` gebruiken. Zodra er iets in de pool geschreven wordt, kan er uit gelezen worden. Echter, mocht de taak niet op tijd zijn met het lezen van de Pool, bestaat er de kans dat een andere taak die in deze pool schrijft, de er al in staande data overschrijft.

Als we gaan kijken naar andere RTOS-en zien we ook weer een pool terug. Dit wordt weliswaar iets anders genoemd, namelijk "Memory Pool", hetzelfde als wat "Channel" in het HU-RTOS is. Er kan in geschreven worden en gedraagt zich als een "Queue", volgens het FIFO (First In First Out) principe.

"Memory pools are basically just memory you've allocated in advance (and typically in big blocks). For example, you might allocate 4 kilobytes of memory in advance. When a client requests 64 bytes of memory, you just hand them a pointer to an unused space in that memory pool for them to read and write whatever they want. When the client is done, you can just mark that section of memory as being unused again."[5]

3 Deelvraag 2

3.1 Toelichting

"Welke open source RTOS-en zijn beschikbaar?"
Er zijn een heel aantal RTOS-en beschikbaar voor Arduino. Dit onderzoek zal er echter twee bespreken, waaronder FreeRTOS, en Mbed OS5. Deze RTOS-en zijn gekozen omdat er duidelijke documentatie beschikbaar is en omdat de concurrency mechanismen het meest met het HU-RTOS overeenkomt.

3.2 FreeRTOS

FreeRTOS is een open-source RTOS die in C is geschreven, de voorganger van C++. Echter is er een C++ wrapper omheen geschreven door Micheal Becker. Deze wrapper handelt C functies af via C++ code. Het mag dus duidelijk wezen dat er zowel een C, als een C++ versie is. Het RTOS is beschikbaar voor het ARM platform. Ook is er documentatie te vinden over het RTOS, die kun je hier vinden. Als je op de link klikt van Michael Becker, zul je zien dat de laatste push vrij recent was. Je kunt dus stellen dat het nog steeds regelmatig wordt geüpdatet.[2]

Op 22 juli 2016 schreef Michael Becker:

"For anyone who may be interested, I've created an object oriented interface on top of FreeRTOS using C++ classes that encapsulate most of the fundamental APIs. They allow you to craft a project using C++ code and conventions while handling the interface logic between the C and C++ code for you."

Waarop Javier antwoordde:

"Very interesting and amazing job. I'm taking a look into it, but as embedded system programmer my first concern is about size in RAM and Flash for ARM Cortex cores."

Wat dus bewijst dat het FreeRTOS geschikt is voor zowel ARM processoren als de programmeertaal C++. In deze link zul je zien, dat het inderdaad het ARM platform ondersteunt.

3.3 Mbed OS5

Mbed OS is een open-source RTOS net als FreeRTOS. Deze RTOS is compatibel met C/C++ compilers en draait op veel platformen.[3] De laatste release van Mbed OS5 was 5.10.2. Deze werd op 24 oktober 2018 gepubliceerd. Uiteraard beschikt

Mbed OS5 over documentatie over de vele functionaliteiten die deze API biedt. Deze documentatie is hier terug te vinden[4].
Mbed OS5 biedt veel meer dan alleen een RTOS, bijvoorbeeld:

- driver API's, bijvoorbeeld voor het lezen/schrijven van digitale/analoge pinnen;
- NFC reader hardware communicatie API;
- Bluetooth API;
- verschillende netwerk interface APIs

3.4 StateOS

Ook het StateOS is net als de vorige genoemde RTOS-en een open source RTOS. StateOS is idem geschikt voor ARM processoren. Verder bevat StateOS timers, flags, events, mutexes, pools en queues.[6]De meest recente update was op 2 november 2018 (gezien op 5 november 2018). Ook is dit OS zeer eenvoudig in gebruik, snel en lijkt het sterk op het huidige HU-RTOS. Een zeer geschikte kandidaat voor deelvraag 3.

4 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?"

4.1 Toelichting

Als we kijken tussen alle beschikbare RTOS-en zullen we zien dat er veel RTOS-en zijn die veel meer bieden dan het HU-RTOS. Het is een RTOS dat veel op het HU-RTOS lijkt. Dit is StateOS. StateOS wordt als volgt omschreven:

"Free, extremely simple, amazingly tiny and very fast real-time operating system (RTOS) designed for deeply embedded applications. Target: ARM Cortex-M, STM8. It was inspired by the concept of a state machine. Procedure executed by the task (task state) doesn't have to be noreturn-type. It will be executed into an infinite loop. There's a dedicated function for immediate change the task state."[6]

Bij deze library moet er echter wel veel gebruik gemaakt worden van lambda functies. Lambda functies zijn een aantal instructies die je meegeeft aan een andere functie, die hem dan pas uitvoert. Er zijn, net zoals in het HU-RTOS, Event Flags. Deze library bevat een `flg_wait(<flag_t>)`, vergelijkbaar met de `wait(<waitable>)` functie die wordt overgeërfd van `rtos::task<>`. Er kan dus worden gewacht op Event Flags.
Het StateOS biedt ook timers, maar geen clock. Een timer kan wel worden gebruikt als een clock, door eerst een bepaalde waarde aan de timer te geven en vervolgens op de timer te wachten.
Verder biedt het StateOS mutexes, queues, mailboxes en tasks net zoals in HU-RTOS. De documentatie is terug te vinden in de referenties bron link.

4.2 Conclusie

In vergelijking tot FreeRTOS en MbedOS5, heeft StateOS de meeste overeenkomsten met de huidige concurrency mechanismen van het HU-RTOS. De taken worden op veel punten dezelfde wijze geïmplementeerd.

5 Deelvraag 4

"Welke mechanismen van het Arduino RTOS worden niet ondersteund door de beschikbare RTOS-en?"

5.1 Toelichting

Om van deze vraag een goed beeld te krijgen, zullen we in dit onderzoek eerst een tabel maken van de zojuist omgeschreven RTOS-en. Deze tabel geeft een overzicht van de ondersteunde synchronisatiemechanismen. Dit is aan de hand van eerder toegelichte bronnen onderzocht.

RTOS:	**HU-RTOS**	**StateOS**	**FreeRTOS**	**MbedOS5**
Flag	Ja	Ja	Nee	Ja
Queue/channel	Ja	Ja	Ja	Ja
Pool	Ja	Ja	Ja	Ja
Mutex	Ja	Ja	Ja	Ja
Timer	Ja	Ja	Ja	Ja
Clock	Ja	Ja	Nee	Nee
Multiple event waiting	Ja	Nee	Nee	Nee

5.2 Conclusie

Zoals in het tabel te zien is, komt het StateOS het meest overeen met HU-RTOS. StateOS heeft een clock. Die wordt in de documentatie "tick-less mode" genoemd. Uit deze tabel blijkt dat geen van de onderzochte RTOS-en multiple event waiting ondersteunt.
MbedOS5 heeft, in tegenstelling tot StateOS en FreeRTOS, een grote hoeveelheid aan API's die niet altijd nodig zullen zijn voor een eenvoudige Arduino applicatie. StateOS is beknopter, makkelijk te leren en snel in gebruik.
Als de applicatie uitgebreider is die veel hardware moet aansturen, is een keuze voor MbedOS5 over StateOS plausibel. Dit is vanwege het feit dat Mbedos5 veel drivers heeft en veel IoT (Internet of Things) oplossingen biedt.

6 Deelvraag 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?"

6.1 Toelichting

6.1.1 Realisatie Clock

De niet ondersteunde mechanismen van het StateOS en MbedOS5 zijn het multiple event waiting en de clock. In plaats van een clock kan de gebruiker een timer gebruiken. De gebruiker dient dan bewust, tijdens het programmeren, de timer opnieuw te setten en hier vervolgens op te wachten. De werking van deze implementatie werkt exact hetzelfde als een clock.

6.1.2 Realisatie Multiple event waiting

Een manier om multiple event waiting te realiseren is door middel van "Variadic Functions"[7]. Variadic functions zijn functies, waarmee je "oneindig veel" parameters kan meegeven. In ons geval zou je dus een wait functie kunnen maken met als parameters, `waitable`, om zo vervolgens over iedere waitable te itereren, en de eerste te `return`-en, die optreedt. Over het algemeen ziet een simpele Variadic Function als het volgt uit.

```
#include <iostream>
#include <cstdarg>

void simple_printf(const char* fmt...) {
   va_list args;
   va_start(args, fmt);
   while (*fmt != '\0') {
      if (*fmt == 'd') {
         int i = va_arg(args, int);
         std::cout << i << '\n';
      } else if (*fmt == 'c') {
         // note automatic conversion to integral type
         int c = va_arg(args, int);
         std::cout << static_cast<char>(c) << '\n';
      } else if (*fmt == 'f') {
      double d = va_arg(args, double);
      std::cout << d << '\n';
      }
      ++fmt;
   }
   va_end(args);
}

int main() {
   simple_printf("dcff", 3, 'a', 1.999, 42.5);
}
```

Met als output:

```
3
a
1.999
42.5
```

7 Conclusie

Om terug te komen op de hoofdvraag:

"Met behulp van welk Open Source Realtime 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?"

Zoals eerder besproken bleek tijdens het onderzoek dat er twee RTOS kandidaten voldeden aan onze eisen, namelijk StateOS en Mbed OS5. In ons eerdere onderzoek over MbedOS5, bleek dat er veel drivers en API's waren, die niet altijd nodig zullen zijn, en veel geheugen in beslag nemen. Ook komt de syntax (naamgeving van functies, objecten) niet overeen met dat van het HU-RTOS. In tegenstelling tot MbedOS5 is StateOS een concreet en eenvoudig in gebruik te nemen RTOS.
Concurrency mechanismen worden relatief makkelijk aangemaakt en de GitHub van StateOS biedt veel voorbeelden. Tevens bevat StateOS een clock, dat MbedOS5 niet bevat. Echter is dit eenvoudig te realiseren met een timer.
Uit dit onderzoek blijkt dus dat StateOS het best blijkt te zijn als vervanging voor het HU-RTOS met zo min mogelijk overhead.

8 Referenties

Referenties

[1] Wikipedia-bijdragers. (2015, 5 november). Realtimebesturingssysteem - Wikipedia. Geraadpleegd op 17 oktober 2018, van https://nl.wikipedia.org/wiki/Realtimebesturingssysteem

[2] Becker, M. (2018, 20 oktober). michaelbecker/freertos-addons. Geraadpleegd op 24 oktober 2018, van https://github.com/michaelbecker/freertos-addons

[3] Mbed. (z.d.). Open-source RTOS for IoT development | Mbed. Geraadpleegd op 25 oktober 2018, van https://os.mbed.com/

[4] Mbed. (2018, 24 oktober). Open-source RTOS for IoT development | Mbed | Releases. Geraadpleegd op 25 oktober 2018, van https://os.mbed.com/

[5] Stack Overflow (2015, 28 mei). How does memory pools work? [Forumpost]. Geraadpleegd op 5 november 2018, van https://stackoverflow.com/questions/30508183/how-does-memory-pools-work

[6] StateOS. (2018, 19 oktober). StateOS. Geraadpleegd op 2 november 2018, van https://www.osrtos.com/rtos/stateos

[7] C++ Reference. (z.d.). Variadic functions - cppreference.com. Geraadpleegd op 6 november 2018, van https://en.cppreference.com/w/cpp/utility/variadic