

Embedded Realtime OS FreeRTOS auf STM32F4

Michael Ebert
Ad-hoc Networks GmbH
ebert@ad-hoc.network

Christoph Bläßer
Bundesamt für Sicherheit in der
Informationstechnik
christoph.blaesser@gmx.de

Stichwörter

RTOS, ARM, STM32, Real Time.

KURZFASSUNG

Im Rahmen des vorliegenden Papers wird das Echtzeitbetriebssystem FreeRtos vorgestellt. Hierzu werden zu Beginn die allgemeinen Vorgaben für Echtzeitbetriebssysteme beschrieben. Im Verlauf des Textes wird an ausgewählten Beispielen dargestellt, wie FreeRtos diese Anforderungen berücksichtigt und durch geeignete Programmfunktionen umgesetzt.

1. GRUNDLAGEN

1.1 Allgemeine Anforderungen an Betriebssysteme

Betriebssysteme verwalten den Hardwarezugriff und stellen sicher, dass eingesetzte Software die benötigte Rechenzeit zur Verfügung gestellt bekommt. Gleichzeitig regeln Sie den Hardwarezugriff und organisieren den konkurrierenden Zugriff, beispielsweise auf Netzwerkkarten und Festplatten. Sie stellen Funktionen für die Interprozesskommunikation bereit und übernehmen grundlegende Aufgaben wie die Organisation von Arbeitsspeicher.

1.2 Echtzeitsysteme

Mit der steigenden Leistungsfähigkeit von modernen μ Prozessoren, steigen auch die Anforderungen an die Software die auf diese Systeme aufsetzt. Viele dieser Systeme verlangen trotz ihrer Komplexität, dass Teile des Programmablauf in bestimmten zeitlichen Grenzen ausgeführt wird und somit vorhersehbar und deterministisch sind. Systeme die eine solche Anforderung unterliegen werden Echtzeitsysteme genannt. Echtzeitsysteme unterliegen einer weiteren Unterteilung in weiche Echtzeitsysteme (soft realtime systems) und harte Echtzeitsysteme (hard realtime systems). Ein weiches Echtzeitsystem soll eine Aufgabe in den vorgegebenen zeitlichen Grenzen ausführen, ein überschreiten ist aber erlaubt und führt nicht unmittelbar zu einem Fehler. Ein hartes Echtzeitsystem hingegen muss die gestellte Aufgabe in den vorgegebenen Grenzen ausführen. Eine Überschreitung macht das System unbrauchbar. Einige Beispielsysteme und deren Echtzeitzuordnung wird in Tabelle 1 gezeigt. Um die grundsätzliche Funktionalität eines Echtzeitbetriebssystems zu erläutern, müssen zu erst die Grundmodelle für den Programmablauf eingebetteter Systeme beschrieben werden. Der Programmablauf eingebetteter Systeme lässt sich auf drei Modelle zurückführen (Abbildung 1). Eingebettete Anwendungen können in einer einzigen Schleife (mit oder ohne Interrupt Unterbrechungen) laufen oder aber in event-gesteuerten nebenläufigen eigenständigen Programmabschnitten (Thread oder Task¹)

¹Nachfolgenden wird Task benutzt, da dies der geläufige Begriff bei FreeRtos ist. In der Literatur zu Echtzeitsystemen ist der Begriff nicht exakt definiert.

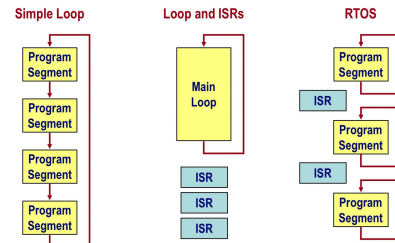


Abbildung 1. Übersicht Programmabläufe
Quelle: <http://www.embedded.com/>

ausgeführt werden. Die nebenläufige Ausführung der unterschiedlichen Programmsegmente ist nur durch einen RTOS Kernel (Scheduler) zu erreichen. Ein RTOS Kernel abstrahiert Timing Informationen und kümmert sich darum, dass die nächste Task rechtzeitig ausgeführt wird. Der Entwickler ist dafür verantwortlich, dass die Task die gewünschte Aufgabe im zeitlichen Rahmen ausführt. Wie sichergestellt werden kann, dass eine Task harten oder weichen Echtzeitanforderungen entspricht wird Abschnitt 4 beschrieben. Für viele kleine Anwendungen kann die Nutzung einer einzigen Schleife durchaus sinnvoll sein, sollten beispielsweise die Ressourcen so knapp sein, dass ein Overhead an Funktionalität ausgeschlossen werden muss. Ein großer Nachteil der „einschleifen Variante“ ist die permanente Nutzung des Prozessors. Besonders bei akkubetriebenen Geräten wie IoT Devices oder Mobiltelefonen wird sehr genau auf die Energieaufnahme geachtet. Ein RTOS bietet hingegen Funktionen mit denen sehr leicht ermittelt werden kann, ob ein Gerät in einen Schlafmodus wechseln kann, dies wird in Abschnitt 2.11 an Beispielen von FreeRtos und einem ARM μ Prozessor demonstriert. Neben der Echtzeitfähigkeit gibt es aber noch viele weitere Vorzüge für den Einsatz eines Echtzeitbetriebssystems. Durch das Herunterbrechen der Anwendung in Task entstehen viele kleine Module, die jeweils eine kleine Teilaufgabe des Gesamtsystems übernehmen. Durch ein sauber definiertes Interface zur Kommunikation der Tasks, lässt sich die Entwicklungsarbeit leicht auf mehrere Entwickler-Teams verteilen. Dies ermöglicht auch den Einsatz von agilen Entwicklungsmethoden wie Scrum in der Entwicklung von eingebetteten Systemen.

TODO Überleitung FreeRtos

1.3 Anforderungen an Echtzeitbetriebssysteme

Echtzeitbetriebssysteme kommen zum Einsatz wenn neben den oben genannten Anforderungen an ein normales Betriebssystem weitere Anforderungen gestellt werden, die ein normales Betriebssystem nicht berücksichtigt. griff nicht exakt definiert.

Beispiel	Echtzeit Typ
Tastatur Controller	Soft Realtime
Echtzeit Media Streaming	Soft Realtime
Controller CD Laufwerk	Hard Realtime
Airbag System	Hard Realtime

Tabelle 1. Beispiele Echtzeitsystem

Dies können beispielsweise garantiert berechenbare Reaktionszeiten sein wie sie in der Fabrikation oder im Automobilbereich gefordert werden oder geringe Leistungsaufnahmen wie bei Komponenten des Internet of Things (IoT). Insgesamt wird zwischen Harten und Weichen Echtzeitkriterien unterschieden. Diese Gliedern sich wie folgt:

- Antwortzeiten
- Start-zu-Start Jitter von Tasks und Software-Komponenten
- Mehrfachaktivierung von Tasks
- Ausführungsreihenfolge von Software-Komponenten
- Latenzen von Wirkketten (Event Chains)
- Verlust von Daten
- Alter der Daten
- Datenkonsistenz
- Wiederverwendung von Daten
- Verlust von Interrupts
- Blockierung von Interrupts

Aufgrund der Eingangs geschilderten Einsatzbereiche ist leicht zu erkennen, dass Echtzeitbetriebssysteme häufig in Umgebungen zum Einsatz kommen, in denen besondere Anforderungen an die Hardware gestellt werden. Häufig verfügt die Hardware nur über begrenzte Speicherkapazitäten, über geringe Wärmeableitfähigkeiten und damit geringe Rechenleistung. Die zur Verfügung stehende Energie muss bei der Entwicklung ebenfalls berücksichtigt werden. Vor diesem Hintergrund benötigen Echtzeitbetriebssysteme nur wenig Speicherplatz und implementieren Funktionen um den Prozessor und die angeschlossene Peripherie nur kurzzeitig zu belasten und in der restlichen Zeit in den Ruhezustand zu versetzen.

2. FREERTOS

2.1 Geschichte

2.2 Zielsysteme STM32F4 (ARM Cortex M3)

2.3 Entwicklungsumgebung

FreeRtos ist im Prinzip nicht an eine spezielle Entwicklungsumgebung gebunden. Bevor eine Entwicklung beginnt ist es dennoch ratsam sich einen Überblick über die verfügbaren IDEs zu machen. Mit steigender Anzahl an komplexen Task und einer erhöhten Anzahl an Kommunikation zwischen den Tasks, steigt auch die Komplexität des Debuggens solcher Anwendungen. Viele IDEs bieten spezielle Plugins die einem Entwickler dabei helfen den Ablauf nachzuvollziehen. Folgende Erweiterungen werden für die meisten IDEs angeboten Thread Awareness, Rtos Awareness. Entwicklungsumgebungen müssen grundsätzlich in zwei

Kategorien eingeteilt werden: proprietäre und nicht proprietäre. Die proprietären Entwicklungsumgebungen bieten den Vorteil der nahtlosen Einbindungen von uProcessoren und Echtzeitbetriebssystem. Bei der Entwicklung von ARM uProcessoren sind hier Keil (Arm), IAR Workbench und True Studio (Atollic) zu nennen. Diese Entwicklungsumgebungen lassen sich zum Teil auf frei verwenden, allerdings mit starken Einschränkungen. Auf der nicht proprietären Seiten steht im Prinzip nur Eclipse CDT zur Verfügung, es ist komplett frei in der Verwendung und hat keine Beschränkungen wie z.B. maximal Codesize. Nachteil ist hier das die Integration nicht so einfach ist wie bei den anderen System, es stehen Beispielsweise keine Beispiel Projekte für Eclipse und FreeRtos zur Verfügung, stattdessen muss alles eigenständig installiert und konfiguriert werden.

2.4 Ordner und Dateistrukturen

2.5 Einrichten und Konfiguration

2.6 Scheduling

2.7 Memory Management

2.8 Interprozess Kommunikation

2.9 Software Timer

2.10 Interrupt Handling

2.11 Low Power Modes auf Stm32F4

3. MAßNAHMEN ZUR SICHERSTELLUNG DER ECHTZEITFUNKTIONALITÄT

4. ECHTZEITANALYSE

5. DEBUGGING VON ECHTZEITSYSTEMEN

6. ECHTZEIT PATTERN

7. KOMPLEXITÄT DURCH NEBENLÄUFIGKEIT

8. ZUSAMMENFASSUNG

Literatur