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 FreeRos diese Anforderungen berücksichtigt und durch geeignete Programmfunktionen umsetzt.

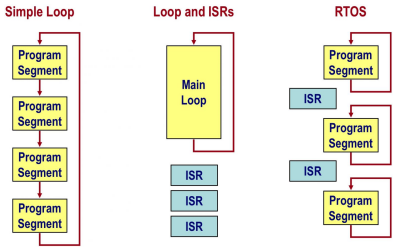

Abbildung 1. Übersicht Programmabläufe
Quelle: `http://www.embedded.com/`

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 organiseren den konkurierenden Zugriff, beispielsweise auf Netzwerkkarten und Festplatten. Sie stellen Funktionen für die Interprozesskommunikation bereit udn 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 und harte Echtzeitsysteme (soft / hard realtime systems). Ein weiches Echtzeitsysteme soll eine Aufgabe in den vorgegeben 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. Der Programmablauf eingebetteter Systeme lässt sich auf drei Grundmodelle zurückführen, siehe Abbildung 1. Eingebettete Anwendungen können in einer einzigen Schleife (mit und ohne Interrupt Unterbrechungen) laufen oder aber in event-gesteuerten nebenläufigen eigenständigen Programmabschnitten (Thread oder Task) ausgeführt werden. Diese Nebenläufigkeit ist nur durch einen RTOS Kernel (Scheduler) möglich. 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 Task harten bzw. 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 Device 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 3.2 an Beispielen von FreeRtos und einem ARM μProzessor demonstriert. Neben der Echtzeitfähig 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. Die Anwendungen werden so skalierbar und erweiterbar. Die Entwicklungsarbeit lässt sich leicht auf Teams verteilen, die unabhängig ihre Module (Tasks) entwickeln und testen. 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. 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 leich zu erkennen, dass Echzeitbetriebssysteme häufigin 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 verseten.