Intertask Kommunikation in Echtzeitbetriebsystemen

Stefan Lindörfer

Seminar: Embedded Systems
Prof. Dr. Reiner Kolla
Wintersemester 2020/21

Abstract Die Informationsübermittlung zwischen Tasks in Echtzeitbetriebsystemen ist von fundamentaler Bedeutung und kann je nach Anforderungsszenario auf verschiedene Arten erfolgen. In der vorliegenden Arbeit wird zunächst ein Überblick über die verschiedenen Möglichkeiten gegeben sowie darauffolgend jeweils vertieft auf die Mechaniken eingegangen und durch Beispiele demonstriert.

1 Einführung

Echtzeitbetriebsysteme (auch **R**eal **T**ime **O**perating **S**ystems, kurz RTOS genannt) sind aus der Informatik nicht mehr wegzudenken und begegnen uns oft unbewusst im Alltag in zahlreich verschiedenen Anwendungsbereichen. So werden sie etwa in modernen Automobilen als Betriebssysteme eingesetzt, zur Steuerung und Regelung industrieller Anlagen verwendet oder im Verkehrswesen (z.B. Schienenverkehr- oder Ampelsteuerungen) genutzt [vgl. 1, S. 157]. Auch als Betriebssysteme von Satelliten in der Raumfahrt sowie für den Einsatz in Flugzeugen sind sie geeignet [2].

Die Besonderheit dieser Art von Betriebssystemen ist die Fähigkeit, Echtzeit-Anforderungen umsetzen zu können. Das bedeutet (anders als bei Nicht-RTOS) als Softwareentwickler die Zusicherung seitens des Betriebssystems zu besitzen, dass eine bestimmte Aufgabe innerhalb eines vordefinierten Zeitfensters entweder ausgeführt und abgeschlossen (oder unterbrochen) wird. Betriebssysteme anderer Kategorien führen ihre Aufgaben meist schnellstmöglich aus und passen kein Zeitfenster ab, garantieren also keine Echtzeit, sondern setzen auf größtmögliche Performanz.

Aufgaben in Echtzeitbetriebssystemen sind in sogenannte Tasks (oder auch Threads) organisiert, die vom Betriebssystem verwaltet werden. Abhängig vom jeweiligen Szenario können Tasks nicht ausschließlich getrennt voneinander operieren und ihren jeweiligen Aufgaben nachgehen, sondern ein Informationsaustausch zwischen ihnen ist erforderlich: Um etwa einen Ablauf abzubilden, bei dem zuerst Task A und anschließend Task B ausgeführt werden soll, muss eine Information von A nach B übermittelt werden oder, wenn Daten eines Tasks von einem anderen Task zur (synchronisierten) Weiterverarbeitung benötigt werden, ist ein entsprechender Kommunikationskanal erforderlich um die Daten zu transferieren.

Dieser gesamte interne Informationsaustausch wird als Intertask-Kommunikation bezeichnet. Es existieren mehrere allgemeine Möglichkeiten und Techniken, Informationen und Daten zwischen Tasks auszutauschen, abhängig vom jeweiligen Szenario und den Anwendungsanforderungen. Diese Techniken werden im Folgenden nacheinander aufgezeigt und erläutert.

1.1 Überblick

Grundsätzlich unterteilt man Intertask-Kommunikation in drei Kategorien [vgl. 3, S. 79]:

1. **Synchronisation und Koordination von Tasks ohne Datentransfer**
2. **Datentransfer zwischen Tasks ohne Synchronisation**
3. **Datentransfer zwischen Tasks mit Synchronisation**

Punkt 1 enthält anders als 2. und 3. keinen Datentransfer und unterscheidet sich von diesen dadurch, dass lediglich eine Information ausgetauscht bzw. vom Empfänger abgefragt wird, um Tasks zu synchronisieren oder einen Ablauf umzusetzen. Während bei 2. und 3. ein Datentransfer insofern stattfindet, als das Daten ausgetauscht und vom Empfänger für die Weiterverarbeitung genutzt werden, sie also nicht für eine Ablaufsteuerung verwendet werden.

Hier eventuell die Referenzen der jeweiligen Kapitel abbilden? Nützlich? Ist doch eigentlich Job eines Inhalts-

1.2 Begriffsgrundlage

Für eine genauere Betrachtung der einzelnen Kategorien aus 1.1, müssen zunächst die beiden Begriffe Koordination und Synchronisation definiert und pragmatisch voneinander abgegrenzt werden:

- **Koordination**: *„Das Integrieren und Anpassen (einer Reihe von Teilen oder Prozessen), um eine reibungslose Beziehung zueinander herzustellen."* [vgl. 3, S. 80]
- **Synchronisation**: *„Etwas verursachen, bewegen oder ausführen, genau zur exakten Zeit."* [vgl. 3, S. 80]

Es fällt auf, dass die Definition der Koordination keinen Bezug zur Zeit beinhaltet. Der wesentliche Unterschied zwischen Koordinierung und Synchronisierung ist somit der Zeit-Faktor. Während mit einer Koordination ein theoretisch zeitunabhängiger, sequentieller Ablauf von Tasks angestrebt wird, meint Synchronisation dagegen das zeitliche Abgleichen von Vorgängen und legt damit verstärkt Fokus auf die Kerneigenschaft von Echtzeitbetriebssystemen.

2 Task-Interaktion ohne Datentransfer

Müssen, wie schon in 1.1 erwähnt, keine Daten im eigentlichen Sinne zwischen Tasks transferiert werden, sondern nur ein Arbeitsablauf gesteuert werden, spricht man von Task-Interaktion ohne Datentransfer. Hierbei erfolgt lediglich die Übermittlung einer Information von einem Task zum anderen. Dies kann sowohl mit dem Ziel einer Task-Synchronisation durchgeführt werden als auch ohne den bereits festgestellten Zeit-Faktor mittels Koordination, siehe 1.2. Demzufolge wird bei den Möglichkeiten dieser Kategorie auch in diese beiden Fälle unterschieden. Tabelle 1 zeigt diese Unterscheidung auf und gibt gleichzeitig einen Überblick über die jeweils zu verwendeten Konstrukte dieser Kategorie. Wird lediglich Koordination benötigt, werden Condition Flags

Koordination	**Synchronisation**	
Condition Flags	Event Flags	Signale
Operationen:	*Operationen:*	*Operationen:*
`Set`	`Set`	`Wait`
`Clear`	`Clear`	`Send`
`Check`	`Check`	`Check`

Tabelle 1: Koordinierungs- und Synchronisationskonstrukte [vgl. 3, S. 82]

verwendet. Ein Flag ist ein Statusindikator, der einen bestimmten Zustand anzeigt. Ist Synchronisierung erforderlich, dass also der gesteuerte Prozess zeitkritisch auszuführen ist, dann können - je nach Anwendungsfall - Event Flags oder Signale verwendet werden. Alle drei Konstrukte und deren Operationen werden im folgenden genauer erläutert.

2.1 Koordinierung mit Condition Flags

Die einfachste Möglichkeit der Koordination ist das Condition Flag. Tabelle 1 zeigt die auf Condition Flags anwendbaren Operationen (`Set` (setzen), `Clear` (zurücksetzen) und `Check` (überprüfen)).

Abbildung 1 zeigt exemplarisch, wie Condition Flags verwendet werden: Task A übernimmt in diesem Fall die Steuerung des Ablaufs, während Task B darauf wartet, seine Aufgabe auszuführen und laufend überprüft (`Check`) ob das Flag gesetzt wurde. Meistens wird dafür eine globale Boolean-Variable eingesetzt, bei dieser 1 (true) den Zustand `Set` und 0 (false) den Zustand `Clear` repräsentieren kann. Ist es erforderlich, mehrere Zustände einzusetzen, können Aufzählungstypen (enums) verwendet werden: Dieses Vorgehen bietet auch den Vorteil der besseren Lesbarkeit des Quellcodes, da eindeutig verifizierbar ist, welcher Flag-Zustand `Set` bzw. `Clear` darstellt. In kritischeren Situationen, z.B. bei hoher Zugriffsrate auf das Flag (können sich Lese- oder Schreibfehlern

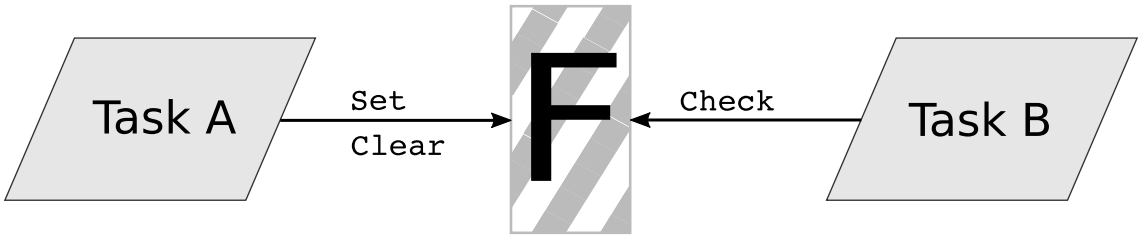

Abbildung 1: Einfache Benutzung von Condition Flags [vgl. 3, S. 83]

```
typedef enum {StartSet, StartReset} StartFlag;
typedef enum {StopSet, StopReset} StopFlag;
```

Abbildung 2: Verwendung mehrerer Flags zur Absicherung und Lesbarkeit [vgl. 3, S. 84–85]

Abbildung 5.5, S. 84

ergeben) oder wenn ein höheres Maß an Ausfallsicherheit gewünscht ist, empfiehlt sich eine Mehrfachabsicherung wie sie Abbildung [] demonstriert: Für jeden Zustand der gesetzt werden kann, existiert ein eigener Statusindikator. Task A übernimmt in diesem Fall ausschließlich das Setzen der Zustände, während Task B zusätzlich beim Überprüfen das Zurücksetzen übernimmt.

Nochmal im Buch lesen!

2.2 Synchronisation über Event Flags

2.3 Synchronisation mittels Signale

3 Datentransfer ohne Synchronisation oder Koordination

Ist ein Datenaustausch zwischen Tasks erforderlich, aber nicht an Koordinierungs- und Synchronisationskriterien gebunden, unterliegt also weder einer zeitlichen Vorgabe noch einem Ablauf, werden verschiedene Datenstrukturen zum Austausch zwischen Tasks verwendet.

3.1 Überblick

3.2 Pools

3.3 Queues

4 Task-Synchronisation mit Datentransfer

Daten, die mit einer zeitlichen Priorität zwischen Tasks ausgetauscht werden, also genau zum richtigen Zeitpunkt beim Empfänger bereit stehen müssen, werden ebenfalls mit einer Datenstruktur transferiert. An diese besteht jedoch ein höherer Anspruch, da sie die Synchronisation sicherstellen muss.

4.1 Mailbox

5 Zusammenfassung

Literatur

[1] Marco Winzker. *Elektronik für Entscheider: Grundwissen für Wirtschaft und Technik*. Vieweg, Wiesbaden, 2008.

[2] Lehrstuhl VIII Universität Würzburg Institute für Informatik. *Rodos - Lehrstuhl für Informatik VIII*. 6. März 2019. URL: `https://www.informatik.uni-wuerzburg.de/aerospaceinfo/wissenschaftforschung/rodos/`.

[3] Jim Cooling. *Real-time Operating Systems: Book 1 - The Theory*. Independently Published, 2017.