Rekonfiguration in Echtzeitsystemen

Adrian Meßner
FAU Erlangen-Nürnberg
adrian.messner@fau.de

ABSTRACT

Viele Echtzeitsysteme umfassen unterschiedliche Betriebsmodi, welche über einen geregelten Moduswechsel eingeleitet werden. Diese Arbeit gibt einen Überblick über mehrere Methoden, um in Echtzeitsystemen einen Moduswechsel durchzuführen. Dabei werden zunächst synchrone Protokolle angesprochen. Sie sind generell leicht zu implementieren, benötigen jedoch eine lange Umstellungszeit. Weiterhin werden asynchrone Protokolle erläutert. Diese kompensieren die Schwächen der synchronen Protokolle, benötigen jedoch eine spezielle Planbarkeitsanalyse. Es wird dabei der Umgang dieser Protokolle mit noch anstehenden Aufgaben aus dem vorherigen Modus erklärt. Zudem wird der Einsatz von Offsets erläutert, um Aufgaben aus dem neuen Modus zu verzögern. Durch diese wird im System temporäre Überlast vermieden.

Keywords

Echtzeitsystem, Moduswechsel, Offset, Protokoll, Betriebsmittel

1. EINFÜHRUNG

Viele technische Systeme mit Echtzeitanwendungen müssen verschiedenste Aufgaben bearbeiten und stehen unter wandelnden Umwelteinflüssen. Daher ist es von Vorteil, wenn diese Echtzeitsysteme in der Lage sind, während der Laufzeit ihr Verhalten für die jeweilige Situation anzupassen und dazu entsprechende Betriebsmodi einzuleiten. Ein anschauliches Beispiel für technische Systeme mit unterschiedlichen Betriebsmodi sind unter anderem Flugzeugsteuersysteme, die je nach Start-, Flug- oder Landephase unterschiedliche Aufgaben und Ziele haben.

Aber auch allgemein häufig eingesetzte Betriebsmodi in Echtzeitsystemen, machen einen Moduswechsel unverzichtbar. Beispiele hierfür sind die Initialisierung des Systems und der Notfallmodus. Dieser dient zur Behandlung extremer Situationen, wie zum Beispiel in den Bereichen des Automobils. Auch zu erwähnen ist der Energiesparmodus, welcher die Arbeitslast zugunsten der Energieversorgung reduziert.

Durch das Einleiten eines Moduswechsels verändert sich der aktuelle Zeitplan eines Echtzeitsystems, wodurch Aufgaben aus beiden Modi in Konflikt geraten und es zur Überlast im System kommen kann. Um einen Wechsel des Betriebsmodus durchzuführen, bedarf es somit eines Protokolls, welches einen gültigen Ablaufplan des Echtzeitsystems sicherstellt.

Zunächst werden in Abschnitt 2 generelle Begriffe zur Thematik erklärt und es wird auf die Anforderungen eines Moduswechsels eingegangen. Eine Klassifizierung der bekannten Protokolle wird in Abschnitt 3 gegeben. Anschließend wird deren Funktionsweise erläutert. Hierzu werden zunächst synchrone und in Anschluss asynchrone Protokolle beschrieben, wobei sowohl periodische als auch nichtperiodische Versionen berücksichtigt werden. Dabei werden Methoden wie das Einsetzen von Offsets beschrieben, um die Aufgaben aus dem neuem Modus zu verzögern. Auch der unterschiedliche Umgang mit Aufgaben aus dem vorherigen Modus wird beschrieben. Weiterhin wird gezeigt wie gemeinsame Betriebsmittel in dieser Zeitspanne behandelt werden können um Datenverletzungen zu vermeiden. Außerdem wird diskutiert wie sehr die einzelnen Techniken die Anforderungen eines Moduswechsels verwirklichen.

2. GRUNDLAGEN

In diesem Abschnitt werden die Grundlagen der Thematik erklärt. Zuerst wird das Auftreten eines Moduswechsels und das Aufgabensystem in Echtzeitsystemen beschrieben. Anschließend wird der Einsatz von Offsets erklärt. In Abschnitt 2.4 werden die Anforderungen an einen Moduswechsel erläutert. Auf die Problematik des Einsatzes von Betriebsmitteln wird im letzten Teilabschnitt eingegangen.

2.1 Betriebsmodi

Ein Moduswechsel wird durch eine Änderung in der Umwelt des Systems, falls zum Beispiel ein Sensor einen kritischen Wert erfasst, oder betriebsintern ausgelöst. Dieser kann zu jedem Zeitpunkt auftreten. Die Anforderung eines Betriebswechsels wird durch eine laufende Aufgabe eingeleitet und als mode change request (MCR) bezeichnet. Dabei wird der aktuelle Betriebsmodus beendet und eine Übergangsphase eingeleitet, welche zum neuen Betriebsmodus überleitet. Ein MCR kann in jedem Betriebsmodus auftreten. Ausnahme ist die Übergangsphase selbst. Die Aufgabe des Betriebssystems ist es daher zu diesem Zeitpunkt andere eingehende MCR entweder zu ignorieren oder einen Mechanismus zu einzuleiten, der diese entsprechend behandelt. Beispielsweise kann hierfür eine Hierarchie innerhalb der Betriebsmodi geschaffen werden [5].

2.2 Aufgaben in Echtzeitsystemen

Ein Echtzeitsystem arbeitet während der Laufzeit mehrere Aufgaben ab, welche in der Regel periodisch gestartet werden. Eine Aufgabe τ_i ist definiert durch eine maximale Ausführungszeit C_i (engl. worst case execution time), einer Periode T_i, einem Termin D_i und einer Priorität P_i. In manchen Echtzeitsystemen darf der Termin einer Aufgabe

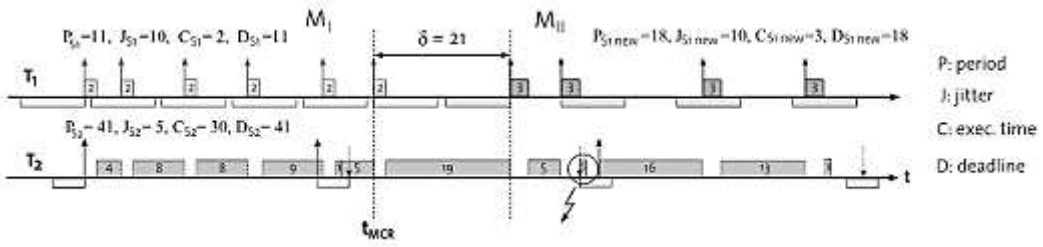

Abbildung 1: Verpasster Termin aufgrund von Überlast

verpasst werden, es wird aber im Folgenden angenommen, dass jeder Termin eingehalten werden muss. Sollte ein Echtzeitsystem nur über einen einzigen Betriebsmodus verfügen, so verändern sich diese Parameter während der gesamten Laufzeit nicht. Falls jedoch neue Betriebsmodi eingeleitet werden können, ist eine Änderung der Parameter von Aufgaben durch einen MCR möglich. Man beschreibt daher die Aufgaben aller Betriebsmodi in Tupeln:

$$\begin{aligned}\tau_i = \Big\{ & \tau_i^{M_1} = \left(C_i^{M_1}, T_i^{M_1}, D_i^{M_1}, P_i^{M_1}\right), \\ & \tau_i^{M_2} = \left(C_i^{M_2}, T_i^{M_2}, D_i^{M_2}, P_i^{M_2}\right), \\ & \cdots, \\ & \tau_i^{M_m} = \left(C_i^{M_m}, T_i^{M_m}, D_i^{M_m}, P_i^{M_m}\right) \Big\}\end{aligned} \tag{1}$$

Es ist möglich, dass manche Aufgaben in einzelnen Betriebsmodi nicht auftreten. Im Tupel wird dabei der Wert $C_i = 0$ gesetzt. Prinzipiell lassen sich nach einem MCR fünf Typen von Aufgaben unterscheiden. Im alten Betriebsmodus gibt es Aufgaben, die bereits vor dem MCR gestartet wurden und noch fertiggestellt werden müssen oder die nach dem MCR abgebrochen werden, da ihre Funktionalität nicht mehr gebraucht wird. Im neuen Betriebsmodus laufen veränderte Aufgaben, die im vorherigen Modus zwar schon existierten, jedoch deren Parameter verändert werden. Beispiele hierfür sind der Termin oder die Periode einer Aufgabe, welche im neuen Modus andere Werte haben können. Weiterhin gibt es unveränderte Aufgaben, die in beiden Betriebsmodi identisch ablaufen, und vollständig neue Aufgaben, die erst mit dem neuen Betriebsmodus eingeleitet werden [5].

2.3 Offsets

Bei einem Moduswechsel wird durch das Einleiten neuer Aufgaben das Erzeugen eines gültigen Ablaufplans erschwert, da diese die Freiräume im Plan füllen. Weiterhin ist es möglich, dass durch die zusätzlichen Aufgaben das System überlastet werden kann. Ein Weg dieses Problem zu lösen ist der Einsatz von Offsets. Durch diese wird die Erstellung eines gültigen Ablaufplans deutlich erleichtert.
Ein Offset ist eine zeitliche Verzögerung für Aufgaben des neuen Betriebsmodus nach einem MCR. Der Wert des Offsets hängt dabei davon ab, von welchen Betriebsmodi der MCR kommt und zu welchem Zeitpunkt er angefordert wird. Hierfür verwendet man für eine Aufgabe τ_i eine Matrix mit den Werten für den Offset Y_i zu den einzelnen Betriebsmodi.

$$\begin{aligned}Y_i = \Big\{ & Y_i^{M_1,M_1}, Y_i^{M_1,M_2}, Y_i^{M_1,M_3}, \cdots, Y_i^{M_1,M_m} \\ & Y_i^{M_2,M_1}, Y_i^{M_2,M_2}, Y_i^{M_2,M_3}, \cdots, Y_i^{M_2,M_m} \\ & \vdots \\ & Y_i^{M_m,M_1}, Y_i^{M_m,M_2}, Y_i^{M_m,M_3}, \cdots, Y_i^{M_m,M_m} \Big\}\end{aligned} \tag{2}$$

Die Zeilen der Matrix bezeichnen dabei den ursprünglichen Modus und die Spalten den zukünftigen Modus. Ein Wert $Y_i(M_a, M_b)$ bezeichnet dabei den Offset einer Aufgabe τ_i von Modus M_a nach M_b. Die Diagonale und einige weitere Einträge, bei denen ein direkter Moduswechsel nicht möglich ist, besitzen den Wert 0 [5].
Eine Herausforderung ist es für jeden Wert dieser Matrix den passenden Wert für das Offset zu ermitteln. Abbildung 1 zeigt beispielhaft zwei periodische Aufgaben, welche statische Prioritäten vergeben wurden, wodurch Aufgabe τ_1 eine höhere Priorität besitzt als Aufgabe τ_2. Hier wurde das Offset so gewählt, dass es die größte Distanz zweier Ankunftszeiten abdeckt. Jedoch zeigt dieses Beispiel, dass die Annahme in diesem Fall alle Termine zu erreichen falsch ist, da mit diesem Offset trotzdem Termine nach einem MCR verpasst werden können [9].

2.4 Anforderungen an einen Moduswechsel

Für einen Moduswechsel müssen bestimmte Anforderungen erfüllt werden. Anhand derer lassen sich später die einzelnen Verfahren evaluieren. Diese Anforderungen sind:

- Planbarkeit: Alle Termine von Aufgaben des alten und des neuen Modus müssen eingehalten werden.
- Periodizität: Einige Aufgaben müssen immer periodisch ausgeführt werden, auch wenn ein Moduswechsel stattfindet. Die periodische Ausführung solcher Aufgaben muss daher gewährleistet werden.
- Unverzüglichkeit: Es können durch einen MCR Aufgaben eingeleitet werden, welche nach einer bestimmten Zeit abgearbeitet sein müssen. Diese Anforderung verlangt schnelle Antwortzeiten nach einem MCR und muss beispielsweise in einem Notfallmodus umgesetzt sein, in dem schnelle Reaktionen wichtig sind.
- Konsistenz: Der Zugriff auf gemeinsame Betriebsmittel muss geregelt erfolgen. Dies ist besonders in der Übergangsphase nach dem MCR eine Herausforderung, da

zu diesem Zeitpunkt Datenupdates erfolgen können und die Prioritäten umgestellt werden. Dadurch sind Protokolle wie das IPCP nicht ausreichend, da diese von statischen Prioritäten ausgehen.

Der Einsatz von Offsets erleichtert die Umsetzung dieser Anforderungen. Sind diese ausreichend groß gewählt, beugt man einer Überlast während der Übergangsphase vor. Auch die Konsistenz wird gewährleistet, da man die Aufgaben des neuen Modus so lange verschieben kann, bis die Prioritätsobergrenzen für die Betriebsmittel angepasst wurden. Andererseits sind Offsets problematisch für unveränderte Aufgaben, da zu große Offsets deren periodische Ausführung unterbinden kann. Große Offsets sind auch für die Unverzüglichkeit ungeeignet, da diese ihr entgegenwirken. Folglich steht man bei der Verwirklichung eines Moduswechsel vor widersprüchlichen Aufgaben. Das Ziel ist es dabei nun ein Protokoll zu entwickeln, das aus diesen Anforderungen den bestmöglichen Kompromiss erzielt [5].

2.5 Betriebsmittel

Für den Fall, dass Aufgaben voneinander abhängig sind und gemeinsame Betriebsmittel benötigen, wird im Folgenden angenommen, dass diese über Shared Memory kommunizieren. Die Vergabe durch Betriebsmittel erfolgt weiterhin über das *priority ceiling protocol (IPCP)* nach Sha et al. [7]. Im IPCP bekommen Betriebsmittel, die von mehreren Aufgaben gefordert werden, Prioritätsobergrenzen zugewiesen. Diese entspricht der höchstprioren Aufgabe, welche das Betriebsmittel benötigt. Eine Aufgabe, der ein freies Betriebsmittel zugewiesen wird, erbt automatisch dessen Priorität. Dadurch werden unkontrollierte Prioritätsumkehr, transitive Blockierungen und Deadlocks vermieden. Da Aufgaben in den einzelnen Betriebsmodi unterschiedliche Prioritäten haben können, können sich folglich auch die Prioritätsobergrenzen der Betriebsmittel ändern. Es wird daher im Folgenden angenommen, dass sich alle Prioritätsobergrenzen nach einem Moduswechsel dynamisch ändern.
Die korrekte Vergabe von Betriebsmitteln muss auch während eines Moduswechsels gewährleistet werden. Im Falle von synchronen Protokollen, in denen Aufgaben aus dem vorherigen und neuen Modus separat abgearbeitet werden, wird die Anpassung der Prioritätsobergrenzen nach einem MCR vollzogen. Bei asynchronen Protokollen, in denen die kombinierte Bearbeitung von Aufgaben aus beiden Modi in der Übergangsphase verwendet wird, ist es schwierig einen passenden Zeitpunkt für die Anpassung der Prioritätsobergrenzen zu finden. Bei zu früher Anpassung kann eine Aufgabe eine Prioritätsobergrenze aus dem neuen Modus erben und somit das IPCP verletzen. Ist die Anpassung zu spät, kann eine Aufgabe aus dem neuen Modus eine alte Prioritätsobergrenze erben.
Es kann durch einen Moduswechsel die Blockierungszeit von neuen Aufgaben erhöht werden, wenn die Prioritätsobergrenzen verringert werden. Ist dabei die Blockierungszeit einer Aufgabe aus dem alten Modus hoch, kann es vorkommen, dass eine Aufgabe aus dem neuen Modus ihren Termin verpasst. Weiterhin ist die Verletzung des IPCP möglich, wenn sich eine Prioritätsobergrenze erhöht und eine neue Aufgabe mit höherer Priorität ein Betriebsmittel beanspruchen möchte, welches noch nicht angepasst wurde. Die Verletzung des Protokolls erfolgt dadurch, dass es keine Aufgaben geben darf, die ein Betriebsmittel verwenden und eine höhere Priorität als die Prioritätsobergrenze besitzen. Dieser Fall kann jedoch durch den Einsatz von Offsets vermieden werden.
In einigen asynchronen Protokollen werden daher globale Prioritätsobergrenzen (engl.: *ceiling of ceilings*) verwendet. Eine globale Prioritätsobergrenze ist die höchste Prioritätsobergrenze, welche ein gemeinsames Betriebsmittel in allen Modi besitzt. Diese wird bei der Initialisierung des Systems einem Betriebsmittel zugewiesen und während der Laufzeit nicht mehr verändert. Dadurch müssen die Prioritätsobergrenzen bei einem MCR nicht geändert werden. Dies kann jedoch die Planbarkeitsanalyse des Systems beeinträchtigen. Es können alte Aufgaben, die ein Betriebsmittel mit einer geringen Prioritätsobergrenze im alten Modus besitzen, während der Übergangsphase hochpriore Aufgaben blockieren. Dieses Ereignis kann auftreten, falls die globale Prioritätsobergrenze hoch genug ist. Dieser Fall wird auch als exzessives Blockierungsproblem (engl.: *excessive blocking problem*) bezeichnet. Real und Wellings haben hierfür zwei Wege gefunden dieses Problem zu lösen [6]: Eine Lösung ist beispielsweise das Ändern der Prioritäten der Aufgaben. Dadurch wird die Anzahl der blockierenden Aufgaben verringert, erfordert jedoch möglicherweise eine höhere Anzahl an Prioritätsebenen. Eine weitere Methode ist das Modifizieren der Interfaces für gemeinsame Betriebsmittel. Dabei wird der Statuts eines Betriebsmittels hinter einem Interface verborgen, welches für jeden Modus eine passende Priorität trägt. Der Nachteil bei dieser Lösung ist jedoch, dass dabei die Anwendung modifiziert werden muss und es zu Overheads kommen kann.

3. PROTOKOLLE FÜR MODUSWECHSEL

Protokolle für Moduswechsel kann man anhand von drei Hauptkriterien unterscheiden. Zum einen besteht die Option Aufgaben des alten Modus nach einem MCR abzubrechen. Das Aktivierungsmuster von unveränderten Aufgaben während der Übergangsphase kann unterschiedlich gestaltet werden und es existiert die Möglichkeit einer kombinierten Ausführung von Aufgaben des alten und neuen Modus während der Übergangsphase.
Mit den Aufgaben des alten Modus nach einem MCR kann man folgendermaßen umgehen:

- Alle laufenden Aufgaben des alten Modus werden sofort abgebrochen. Dies wird vor allem bei einem Notfallmodus angewendet, in dem ein schneller Moduswechsel erforderlich ist. Dies kann jedoch zu Konsistenzproblemen führen.
- Alle noch anstehenden Aufgaben des alten Modus dürfen fertiggestellt werden. Jedoch ist hierbei eine längere Zeitdauer für den Moduswechsel die Folge.
- Nur ein Teil der alten Aufgaben wird abgebrochen. Dies ist die flexibelste Option, jedoch müssen die Aufgaben des alten Modus speziell behandelt werden.

Anhand der Möglichkeiten, wie unveränderte Aufgaben behandelt werden können, werden zwei Arten von Protokollen definiert:

- Periodische Protokolle, in denen unveränderte Aufgaben unabhängig vom Moduswechsel weiter ausgeführt werden
- Nichtperiodische Protokolle, in denen die Ausführung von unveränderten Aufgaben verzögert werden kann, damit Datenkonsistenz gewährleistet wird.

Durch die Möglichkeit Aufgaben des alten und neuen Modus kombiniert auszuführen, kann man weitere zwei Arten von Protokollen unterscheiden:

- Synchrone Protokolle, in denen neue Aufgaben erst ausgeführt werden, wenn alle alten Aufgaben fertiggestellt sind. Diese Protokolle benötigen keine weitere Planbarkeitsanalyse, da keine Überlast geschehen kann. Synchrone Protokolle können sowohl periodisch als auch nichtperiodisch sein.
- Asynchrone Protokolle, in denen in der Übergangsphase die kombinierte Ausführung von Aufgaben des alten und des neuen Modus möglich ist. Diese Protokolle benötigen jedoch eine spezielle Planbarkeitsanalyse.

3.1 Synchrone Protokolle

In diesen Protokollen findet eine getrennte Abarbeitung von Aufgaben aus dem alten und neuen Modus statt. Sie sind generell einfacher gestaltet als asynchrone Protokolle und verursachen in der Regel keine Probleme mit Betriebsmitteln. Die Antwortzeiten für neue Aufgaben sind gegenüber asynchronen Protokollen generell länger.

3.1.1 Idle Time Protocol

In diesem Protokoll nach Tindell und Alonso [10] wird nach einem MCR die Aktivierung von Aufgaben des alten Modus nicht beeinflusst, bis die CPU unausgelastet ist (engl.: *idle instant*). Danach wird die Aktivierung alter Aufgaben beendet und der Moduswechsel gestartet. Die Vorteile dieses Protokolls sind die sehr einfache Implementierung, es bedarf keiner spezielle Planbarkeitsanalyse und es entstehen keine Konflikte mit der Betriebsmittelvergabe, da die Prioritätsobergrenzen im *idle instant* angepasst werden können. Die Nachteile dieses Protokolls liegen in der langen Umstellungsdauer, die vor allem besonders groß ist, wenn zuvor alle Aufgaben des alten Modus zuvor gleichzeitig ausgelöst wurden (engl.: *critical instant*).

3.1.2 Single Offset Protokolle

Diese Art von Protokollen verzögert alle Aufgaben des neuen Modus nach einem MCR um ein Offset. Je nach Behandlung der unveränderten Aufgaben verhalten diese sich periodisch oder nichtperiodisch.

Maximum-period offset: Alle Aufgaben des neuen Modus werden in diesem Protokoll nach Bailey [1] um die Zeitlänge der Periode der Aufgabe mit der größten Periode aus beiden Modi verzögert. Unveränderte Aufgaben sind dabei nicht betroffen, was dieses Protokoll periodisch macht. Der Vorteil dieses Protokolls liegt darin, dass keine spezielle Planbarkeitsanalyse erforderlich ist. Es hat jedoch den Nachteil, dass die Umstellung auf den neuen Modus viel Zeit benötigt. Um die Konsistenz bei der Benutzung der Betriebsmittel zu gewährleisten werden globale Prioritätsobergrenzen (engl.: *ceiling of ceilings*) verwendet. Dabei werden jedoch die Blockierungszeiten erhöht und die Planbarkeit erschwert.

Minimum single offset (nichtperiodisch): Dieses Protokoll wurde von Real [4] veröffentlicht um die Umstellungszeit des Maximum-period-offset-Protokolls zu verbessern, es ist jedoch nicht periodisch. Nach einem MCR werden alle anstehen Aufgaben aus dem alten Modus fertig gestellt, aber nicht mehr neu ausgelöst. Die Aufgaben des neuen Modus werden um ein Offset Y_i verzögert. Dieser berechnet sich aus der Summe der maximalen Ausführungszeit aller bis dahin auftretenden Aufgaben des alten Modus. Durch die getrennte Abarbeitung alter und neuer Aufgaben entstehen zudem keine Konsistenzprobleme.

Minimum single offset (periodisch): Dieses Protokoll von Real [4] ist eine Modifikation des Vorherigen, in dem unveränderte Aufgaben untergebracht werden und somit Periodizität gewährleistet wird. Der Offset Y_i berücksichtigt neben den ausstehenden Aufgaben des alten Modus die Zeitdauer der periodischen Ausführung unveränderter Aufgaben während der Umstellung. Die Nachteile dieses Protokolls sind jedoch eine langsamere Geschwindigkeit der Umstellung und die Gefahr von der gleichzeitigen Ausführung von Aufgaben aus dem alten und neuen Modus. Dies kann aufgrund der kontinuierlichen Ausführung von unveränderten Aufgaben aus beiden Modi auftreten.

3.2 Asynchrone Protokolle

In diesen Protokollen ist die kombinierte Ausführung von Aufgaben des alten Modus und des neuen Modus zugleich möglich, um kürzere Übergangsphasen zu erhalten. Sie sind besser geeignet, wenn die Anforderung der Unverzüglichkeit im Vordergrund steht. Aufgrund der zeitgleichen Ausführung von Aufgaben aller Arten ist eine spezielle Planbarkeitsanalyse notwendig.

3.2.1 Utilization Based Protocol

Das Protokoll wurde von Sha et al. [8] veröffentlicht und setzt periodische Aufgaben nach dem Rate Monotonic Verfahren voraus. Das Protokoll orientiert sich an der frei verfügbaren Prozessorkapazität und den Prioritätsobergrenzen. Durch das Entfernen oder Zurückstellen einer Aufgabe aus dem alten Modus wird frei verfügbare Prozessorkapazität gewonnen, welche für das Einführen von neuen Aufgaben genutzt werden kann. Das Protokoll benötigt daher eine dynamische Überwachung des Prozessors um die verfügbare Prozessorkapazität zu berechnen. Das Ziel ist daher die Aufgaben des neuen Modus zu verzögern, um Datenkonsistenz zu gewinnen. Tindell et al. [11] haben jedoch bewiesen, dass das Protokoll unzureichend ist. Es wird zwar das Auftreten eines *critical instant* berücksichtigt, dieser muss jedoch nicht notwendigerweise die längst mögliche Ausführung für einen Moduswechsel bedeuten.

3.2.2 Asynchrones Protokoll mit Periodizität

Dieses Protokoll wurde von Tindell et al. [11] für Systeme, die nach dem Deadline Monotonic Verfahren arbeiten, entworfen. Das Protokoll geht folgendermaßen vor:

- Aufgaben die im alten Modus ausgelöst wurden und

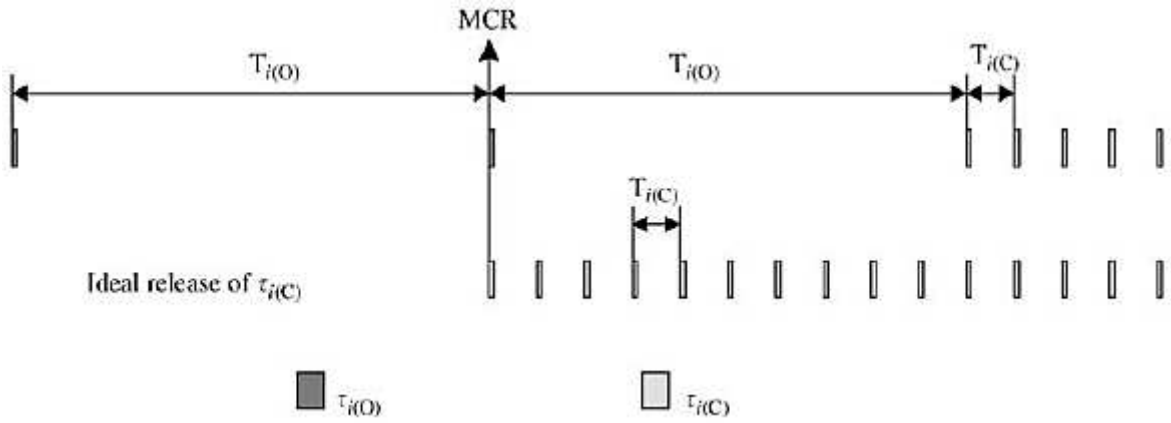

Abbildung 2: Eine Aufgabe τ_i verpasst nach dem Moduswechsel ihre Termine

noch nicht beendet sind, werden fertig gestellt und werden nach einem MCR nicht mehr neu gestartet. Es werden somit keine Aufgaben abgebrochen.

- Unveränderte Aufgaben werden unabhängig vom Moduswechsel weiter ausgeführt
- Aufgaben, deren Parameter aus dem alten Modus verändert wurden, werden direkt am Ende der Periode ihrer alten Version ausgelöst.
- Komplett neue Aufgaben werden nach einem Offset relativ zum MCR eingeführt.

Das Protokoll kann mit Aufgaben umgehen deren WCET sich ändert, jedoch ist die Analyse pessimistisch. In ihr wird zunächst nach Aufgaben aus dem vorherigen Modus, nach unveränderten, veränderten, und komplett neuen Aufgaben unterschieden. Um die Durchführungszeit zu ermitteln werden Zeitfenster verwendet. Eine Aufgabe τ_i aus den alten Modus kann hier durch zwei verschiedene Arten verzögert werden. In einem Fall kann die Aufgabe bei vor einem MCR gleichzeitig mit anderen höherprioren Aufgaben ausgelöst werden, im anderen Fall wurde die Aufgabe zuvor gestartet und es werden höherpriore Aufgaben zeitgleich mit dem MCR ausgelöst. Für die Antwortzeit der Aufgabe wird das Maximum aus den beiden Fällen berücksichtigt. Bei der Analyse der Aufgaben im neuen Modus werden drei weitere Fenster verwendet, welche die Ausführung von τ_i berücksichtigen und komplett neue Aufgaben miteinbeziehen.
Das Protokoll ist somit für unveränderte Aufgaben geeignet, da es periodisch ist. Es ist jedoch weniger für Aufgaben geeignet, deren Periode nach dem Moduswechsel stark verkürzt wird. Falls eine solche Aufgabe kurz nach einem MCR ausgelöst wurde, wird sie noch nach dem vorherigen Modus fertig gestellt bis dessen Periode beendet ist. Erst danach wird sie im neuen Modus mit der verkürzten Periode ausgeführt. Dadurch kann die Aufgabe einige Aktivierungen verpassen. Abbildung 2 zeigt beispielhaft eine Aufgabe die direkt vor einem MCR ausgelöst wird und deren Periode sich im neuen Modus verzehnfacht. Diese kann im schlimmsten Fall zehn mal ihren Termin verpassen. Zudem können in diesem Protokoll nur komplett neue Aufgaben durch ein Offset verzögert werden und es werden globale Prioritätsobergrenzen für gemeinsame Betriebsmittel verwendet, wodurch die Planbarkeit während der Übergangsphase erschwert wird.

3.2.3 Nichtperiodisches Asynchrones Protokoll

Ein periodisches asynchrones Protokoll wurde 1998 von Pedro und Burns [3] [2] veröffentlicht. Im Gegensatz zu den vorherigen Protokollen, können in diesem alle Aufgaben des neuen Modus über ein Offset verzögert werden, wodurch es nicht periodisch ist. Dadurch ergeben sich folgende Vorteile:

- Für jede Übergangsphase kann immer eine korrekte Planbarkeitsanalyse durchgeführt werden.
- Da jede Aufgabe verzögert werden kann, ist es immer möglich einen Zeitpunkt zu finden, in dem die Prioritätsobergrenzen für die Betriebsmittel angepasst werden können, wodurch keine Konsistenz-Probleme entstehen.
- Die Planbarkeitsanalyse wird durch einen definierten Startpunkt für alle Aufgaben des neuen Modus deutlich vereinfacht.
- Es besteht die Option Aufgaben aus dem vorherigen Modus abzubrechen.

Aus diesen Ansätzen ergeben sich jedoch auch folgende Nachteile:

- Die Planbarkeitsanalyse dieses Protokolls teilt sämtliche Aufgaben in die in Abschnitt 3 gezeigten Gruppen und berechnet für jede einzelne die größtmögliche Durchführungszeit. Die Analyse für unveränderte Aufgaben erfolgt jedoch pessimistisch, obwohl sie unabhängig von den anderen Aufgaben ausgeführt werden sollten. Es kann jedoch der Fall auftreten, dass eine solche Aufgabe zweimal in ihrer Periode ausgeführt wird, wie Abbildung 3 zeigt. Dieser ereignet sich, wenn die Aufgabe im alten Modus kurz vor dem MCR ausgelöst wurde und ohne Offset direkt danach nochmal neu eingeführt wird. In diesem Protokoll wird somit die Ausführungszeit in einer Periode zweimal berechnet, und zwar jeweils einmal für beide Modi.

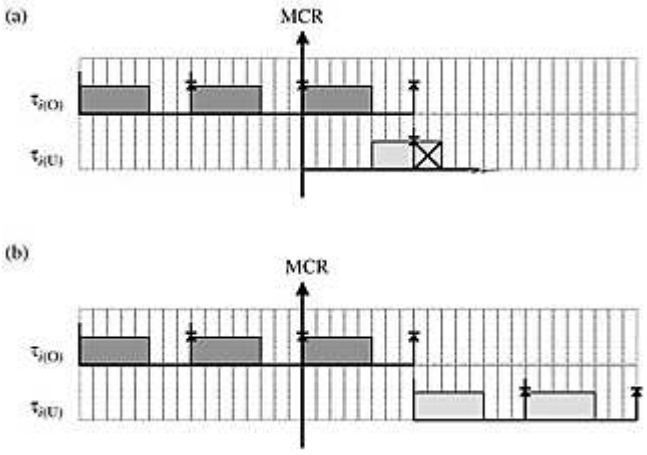

Abbildung 3: (a) die pessimistische Analyse einer unveränderten Aufgabe, (b) die gewünschte Analyse

- Es wird angenommen, dass alle Aufgaben unabhängig sind, wodurch Blockierungszeiten nicht berücksichtigt werden.
- Das Protokoll liefert keinen Algorithmus, in dem eine exakte Berechnung von Offsets gegeben ist.

3.2.4 Asynchrones Protokoll mit zwei Offsets

Real und Crespo [5] veröffentlichten 2004 ein weiteres asynchrones Protokoll. In diesem werden nach einem MCR begonnene Aufgaben aus dem vorherigen Modus, fertiggestellt oder optional sofort abgebrochen. Veränderte Aufgaben werden nach einem Offset Y_i nach dem MCR ausgelöst. Vollständig neue Aufgaben werden nach dem selben Offset eingeführt. Unveränderte Aufgaben werden standardmäßig ohne Offset nach Ende ihrer Periode im vorherigen Modus eingesetzt. Sollte die Planbarkeit dieser Aufgaben in der Übergangsphase nicht möglich sein, so werden diese nach einem Offset Z_i relativ nach ihrer Periode im alten Modus gestartet. Das Protokoll ist damit asynchron und periodisch mit der Option nichtperiodisch zu werden.
Die Offsets müssen lang genug sein einen Ablaufplan zu ermöglichen und sollten Periodizität einhalten. Ein weiteres Kriterium ist, die Offsets für hochpriore Aufgaben zu minimieren. Für dieses Protokoll werden Offsets initial mit einer oberen $Offset_{max}$ und einer unteren Schranke $Offset_{min}$ angegeben. $Offset_{min}$ ergibt sich dabei aus dem Maximum der beiden Werte um einen Ablaufplan und eine Datenkonsistenz zu gewährleisten, $Offset_{max}$ dagegen aus dem Minimum der Offsets um bestmögliche Schnelligkeit und Periodizität zu gewährleisten. Abbildung 4 zeigt den dazugehörigen Pseudo-Code. Zunächst werden drei Vektoren Y_r, Y_{max} und Y_{min} deklariert. Die Vektoren besitzen die Länge von der Anzahl der vorhandenen Aufgaben im neuen Modus. Y_r beinhaltet die Länge der Offsets um Konsistenz zu gewährleisten, Y_{max} die maximalen Offsets, der beispielsweise mit dem Wert der größten Periode aus beiden Modi um einen gültigen Ablaufplan zu sichern, und Y_{min} mit den minimalen Offsets, welcher mit den Werten 0 initialisiert ist. Der Algorithmus sucht in jedem Durchgang gültige Offsets im Intervall $[Y_{min}, Y_{max}]$. Das Minimum stammt dabei aus den

```
1.  Y_r := (0,0,...,0)
2.  Y_max := (∞,∞,...,∞)
3.  Y_min := (0,0,...,0)
4.  loop
5.    ∀i, Y_i := Y_max,i
6.    if Feasible then
7.      Reduce_Offsets
8.      Y'_r := (R_L1,R_L2,...,R_Ln)
9.      exit when Y'_r = Y_r
10.   Y_r = Y'_r
11.   Y_min = Y_r
12.   else
13.     -- Not feasible
14.   end if
15.     -- Feasible
16. end loop
```

Abbildung 4: Algorithmus, um Offset-Werte für Protokoll mit zwei Offsets zu ermitteln

Werten von Y_r. Falls kein gültiger Ablaufplan gefunden werden kann, bricht der Algorithmus ab und erhöht die obere Schranke Y_{max}. Werden reduzierte Offsets gefunden, wird der Vektor Y_r aktualisiert und eine neue Iteration gestartet. Sobald keine neueren Werte ermittelt werden können schließt der Algorithmus ab.

4. FAZIT

In dieser Arbeit wurde eine Übersicht aus Protokollen für Moduswechsel in Echtzeitsystemen gegeben. Es wurden zunächst synchrone Protokolle betrachtet, welche generell einfach gestaltet sind und keine spezifische Planbarkeitsanalyse benötigen. Diese sind dadurch für Systeme geeignet, in denen Periodizität und die Schnelligkeit der Umstellung keine wichtige Rolle spielen. Weiterhin wurden asynchrone Protokolle betrachtet, welche schneller durchgeführt werden können und sich je nach Konfiguration periodisch oder nichtperiodisch verhalten. Da diese jedoch in der Übergangsphase kombiniert Aufgaben aus beiden Modi bearbeiten können, ist für diese Art von Protokollen eine spezifische Planbarkeitsanalyse notwendig. Am Ende wurde ein spezielles asynchrones Protokoll betrachtet, das für seine Analyse zwei Offsets verwendet und die vorherigen in den Anforderungen an Protokollen zum Moduswechsel übertrifft. Zudem wurde zu diesem Protokoll ein Algorithmus entwickelt, der eine genaue Berechnung der benötigten Offsetwerte liefert.

Literatur

[1] C. M. Bailey. Hard real-time operating system kernel. investigation of mode change. *British Aerospace Systems Ltd.*, 1993.

[2] P. Pedro. Schedulability of mode changes in flexible real-time distributed systems. *University of York, Department of Computer Science*, 1999.

[3] P. Pedro and A. Burns. Schedulabillity analysis for mode changes in real-time systems. *Proceedings of the 10th Euromicro Workshop on Real-Time Systems*, 1998.

[4] J. Real. Mode change protocols for real-time systems. *Universidad Politecnica de Valencia*, 2000.

[5] J. Real and A. Crespo. Mode change protocols for real-time systems: A survery and a new proposal. *Real-Time Systems*, 26(2):161–197, 2004.

[6] J. Real and A. Wellings. The ceiling protocol in multi-moded real-time systems. *Reliable Software Technologies - Ada Europe 99, Lecture Notes in Computer Science*, 1622:275–286, 1999a.

[7] L. Sha and J. B. Goodenough. Real-time scheduling theory and ada. *IEEE Computer*, 23(4):53–62, 1990.

[8] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham. Mode change protocols for priority-driven premptive scheduling. *Real-Time Systems*, 1(3):244–264, 1989.

[9] N. Stoimenov, S. Perathoner, and L. Thiele. Reliable mode changes in real-time systems with fixed priority or edf scheduling. *Computer Engineering and Networks Laboratory ETH Zurich*.

[10] K. Tindell and A. Alonso. A very simple protocol for mode changes in priority preemptive systems. *Universidad Politecnica de Madrid*, 1996.

[11] K. Tindell, A. Burns, and A. Wellings. Mode change in priority pre-emptively scheduled systems. *Proceedings Real Time Systems Symposium*, pages 100–109, 1992.