SICHERHEITSBETRACHTUNG VON EMBEDDED SYSTEMEN ÜBERTRAGEN AUF DAS SCHEDULING-SUBSYSTEM

# Dependability-Betrachtung von Multicore-Scheduling

**Der Embedded Markt stellt sich auf eine neue Herausforderung ein: den Umstieg von Singlecore- auf Multicore-Prozessorsysteme. Dabei soll die Umsetzung der Norm ISO 26262 die Funktionale Sicherheit der elektrischen und elektronischen Systeme im Kraftfahrzeug gewährleisten. In diesem Beitrag betrachten die Hochschule Regensburg und die TÜV Süd Automotive GmbH das Scheduling eines Echtzeitsystems als ein sicherheitsrelevantes Sub-System.**

Diskutiert werden im Folgenden Störeinflüsse auf das Scheduling-Subsystem sowie Kenngrößen zur Timing-Error-Bewertung. Ein Online-Reliability-Observer überwacht definierte Grenzbereiche für die Zuverlässigkeit, sodass Error-Behandlungen eingeleitet werden können. Außerdem bewertet das Hochschul-Team anhand der vorgestellten Kenngrößen verschiedene Multicore-Scheduling-Ansätze.

Die Gründe für den Wechsel von Singlecore- oder Multi-Prozessorsystemen auf Multicore-Prozessorsysteme sind bekannt: Durch intelligentere Funktionalität in Steuergeräten steigt der Bedarf an Rechenkapazität, was bei Singlecore-Prozessorsystemen ein erhebliches Problem darstellt. Die Erhöhung der Taktrate, die lange Zeit als performanzsteigernde Maßnahme eingesetzt wurde, ist aufgrund von thermischen Eigenschaften und EMV-Problemen in vielen Bereichen nicht mehr möglich.

## Vorteile von Multicore-Prozessorsystemen

Die parallele Ausführung ermöglicht flexiblere Reaktionen auf äußere Ereignisse. Die Taktrate lässt sich sogar wieder reduzieren, was die oben genannten Probleme verringert. Der größte Vorteil jedoch ist, dass sich die Anzahl der Steuergeräte reduzieren lässt, denn die Rechenleistung von Multicore-Prozessorsystemen ist durch die Anzahl der Kerne skalierbar.

All dies ist nur möglich, wenn man eine auf Multicore-Prozessorsysteme angepasste Softwarearchitektur nutzt und dabei geeignete Scheduling-Algorithmen verwendet. Deshalb sind entsprechende Betriebssystemfunktionen wie Scheduling und Kommunikations-Services näher zu betrachten. Das Scheduling auf Multicore-Prozessorsystemen lässt sich vereinfacht nach der Aufteilungsart des Taskgefüges in zwei Ansätze einteilen: lokales und globales Scheduling (**Bild 1**).

## Lokales Scheduling

Beim lokalen Scheduling, auch als Partitionierung + Singlecore-Scheduling bezeichnet (Bild 1 links), werden Tasks vor der Laufzeit mittels Heuristiken wie BinPacking [1] auf die Kerne eines Multicore-Prozessorsystems aufgeteilt. Während der Laufzeit wird jeder Kern wie ein Singlecore behandelt und bekannte Scheduling-Algorithmen, wie prio-

ritätsbasiertes Scheduling (z. B. OSEK OS) oder Earliest Deadline First (EDF) Scheduling [2], eingesetzt. Der Vorteil des lokalen Scheduling ist, dass der komplexe Prozess der Aufteilung vor der Laufzeit durchführbar ist; Verfahren wurden unter anderem in [1] vorgestellt. Die Komplexität der Algorithmen während der Laufzeit ist gering und somit wenig rechenintensiv. Nebenläufigkeit muss nur für Tasks auf unterschiedlichen Kernen, die auf gemeinsamen Variablen operieren, unterstützt werden. Der Nachteil liegt in der Skalierbarkeit, denn bei Änderungen des Taskgefüges kann eine neue Aufteilung entstehen. Diese ist bei einer Einschränkung der Berücksichtigung von Nebenläufigkeit mit viel Entwicklungsaufwand verbunden.

## Globales Scheduling

Beim globalen Scheduling werden die Tasks während der Laufzeit durch den Scheduler einem Kern zugewiesen, können dort unterbrochen und gegebenenfalls migriert, d. h. auf einem anderen Kern wieder fortgesetzt werden. Hier setzt man Algorithmen wie global EDF, LLREF [3], Partly-Pfair-PD$^2$ [4] und P-ERfair-PD$^2$ [5] ein.

Der Vorteil des globalen Scheduling ist, dass die Systeme hocheffizient betrieben werden können [6], [7], da eine feingranulare Aufteilung möglich ist. Zudem sind diese Lösungen aufgrund der bestehenden Unterstützung von Migration hinsichtlich Änderungen in Taskgefüge und Kernanzahl skalierbar. Der Nachteil liegt in dem erhöhten Scheduling Overhead, der sich aus Scheduling-Laufzeit, Kontext-Wechsel und Migrationskosten zusammensetzt.

Dass diese in der Theorie optimalen Algorithmen auch auf heutige Systeme anwendbar sind, hat in [6] eine Studie im Automotive-Powertrain-Bereich gezeigt.

Ein weiteres wichtiges Unterscheidungsmerkmal des Multicore-Schedulings ist die Vergabe von Prioritäten. Klassische Ansätze vergeben eine feste Priorität (Task-fix Priority Policies), z. B. Rate Monotonic (Prioritätsvergabe entsprechend Aufrufhäufigkeit) oder Deadline Monotonic (Prioritätsvergabe entsprechend relativer Task-Deadline), oder ermöglichen eine Prioritätsänderung zwischen Jobs einer Task (Job-fix Priority Policies), z. B. Earliest Deadline First (Priorität entsprechend absoluter Task-Deadline). Beim dynamischen Scheduling hingegen kann die Priorität auch innerhalb eines Jobs variieren, wie dies bei auch bei den derzeit einzigen als optimal bewiesenen Algorithmen LLREF und der Pfair-Gruppe der Fall ist. Nach dem aktuellen Stand der Wissenschaft deuten die Ergebnisse darauf hin, dass sich Multicore-Prozessorsysteme nur effizient einsetzen lassen, wenn man dynamische Algorithmen verwendet [7].

Entsprechend der Aufteilungsart und der Prioritätsvergabe variieren ebenfalls die Verfahren zur Überprüfung der Echtzeitanforderungen eines Taskgefüges. Für lokales Scheduling mit Task- und Job-fixer Priorität gibt es etablierte Lösungen. Beim globalen Scheduling mit Task-fixer Priorität sind algorithmusspezifische Analysen vorhanden. Beim globalen Scheduling mit dynamischen Prioritäten bieten simulationsbasierte Verfahren [6] eine Möglichkeit zur Echtzeituntersuchung.

## Sicherheitsproblematik des Multicore Scheduling

Sowohl beim lokalen als auch beim globalen Multicore-Scheduling sind die Eigenschaften des Taskgefüges, wie Aktivierungsverhalten $p_i$ und Laufzeiteigenschaften $e_i$, ein Bestandteil des Schedulings. So wird beim lokalen Scheduling beispielsweise anhand des Task-Gewichtes

$$w_i = \frac{e_i}{p_i}$$

eine Partitionierung vorgenommen. Globale Algorithmen, wie P-ERfair-PD$^2$, nutzen das Task-Gewicht zur Bestimmung der Scheduling-Policies.

Die Eigenschaften für das Taskgefüge sind aber oftmals abhängig vom Zustand des Systems. In Singlecore-Prozessorsystemen gibt es wohldefinierte Worst-Case-Szenarien (beim Task- und Job-fixen Scheduling sind dies Worst-Case Execution Time und maximale Aufrufhäufigkeit). Diese sind für dynamisches Scheduling nicht explizit gegeben [8].

Aus diesem Grund muss man die Definition eines verifizierten Systems folgendermaßen erweitern:

**Definition 1**: Ein System besitzt ein „verifiziertes Taskgefüge“ $\tau^v$, wenn bei sämtlichen geplanten Variationen der Eigenschaften des Taskgefüges keine Verletzungen der Echtzeitanforderungen auftreten.

Weicht die Variation des Taskgefüges von der geplanten Variation ab, spricht man von einer Störung (Fault) des Taskgefüges.

**Definition 2**: Ein System besitzt ein „gestörtes Taskgefüge“ $\tau^v + S(\tau)$, wenn der Störeinfluss $S(\tau)$ zu einer Variation der Eigenschaften des Taskgefüges führt, welche von den geplanten Variationen des verifizierten Taskgefüges abweichen.

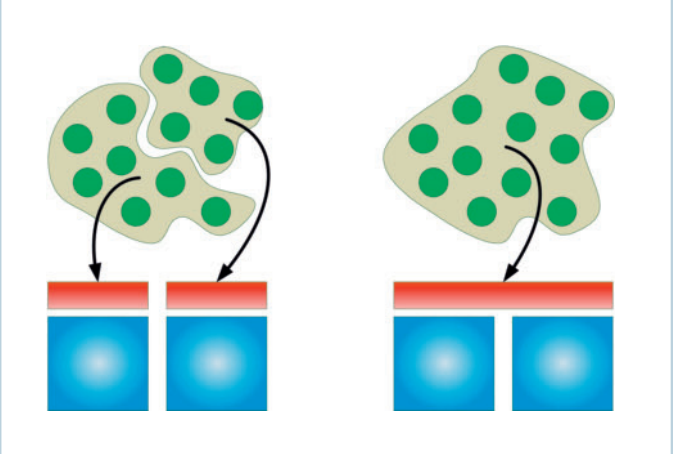

**Bild 1: Schematische Darstellung von lokalem Scheduling (links, Aufteilung der Tasks auf die Kerne vor der Laufzeit; anschließendes Singlecore Scheduling) und globalem Scheduling (Aufteilung der Tasks auf die Kerne zur Laufzeit mittels Scheduling-Algorithmus).**

Das folgende Kapitel behandelt die Art der Störungen näher.

### Störungen des Scheduling-Systems

Um den Einfluss der Störungen $S(\tau)$ auf ein verifiziertes Taskgefüge $\tau^v$ zu untersuchen, ist es nötig, die Störungen zu klassifizieren. Anschließend lassen sich damit störungsabhängige Robustheitsuntersuchungen durchführen.

Explizit vernachlässigt wird hierbei, ob die auf das Scheduling-System wirkenden Störungen durch die Hard- oder Software verursacht werden, da dies für die temporale Bewertung nicht ausschlaggebend ist.

Störungen werden hinsichtlich drei Eigenschaften unterschieden: Störungsart, Intensität und Dauer. Betrachtungsgrundlage ist dabei ein Taskgefüge entsprechend **Bild 2**. Die x-Achse entspricht der Zeit, Task-Aktivierungen sind durch einen Pfeil dargestellt. Im verifizierten Zustand sind Tasks in sequenziell abhängige Task-Sektionen unterteilt (die Unterteilung in Task-Sektionen beruht auf der Modellierung und Untersuchung von Kooperativen Scheduling-Algorithmen, welche nur an Task-Sektionsgrenzen einen Prozessorwechsel ermöglichen). Im Beispiel sind dies zwei Task-Sektionen gleicher Länge. Für das Aktivierungsverhalten wird zur Veranschaulichung von einem periodischen System ausgegangen (die Anwendung auf Jitter-behaftete oder Event-basierte Aktivierungspattern erfolgt analog der jeweiligen Beschreibungsgrößen).

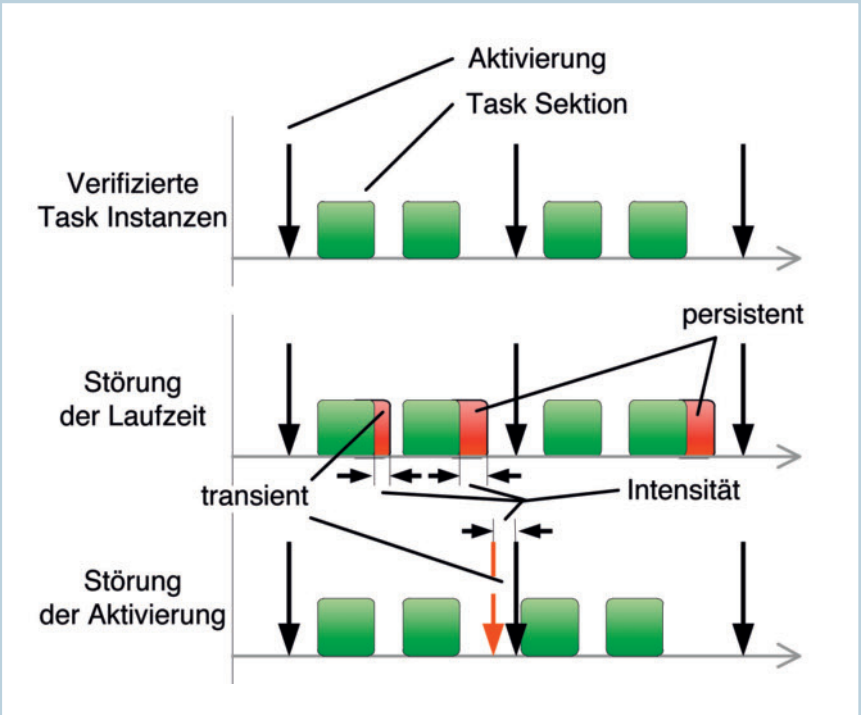


**Bild 2: Mögliche Störeinflüsse auf das Taskgefüge werden hinsichtlich der Störungsart (Laufzeitbedarf, Aktivierungsverhalten), der Intensität und der Dauer (persistent, transient) unterschieden.**

© automotive

- Störungsart: Sie bezeichnet die gestörte Eigenschaft des Taskgefüges. Diese können die Eigenschaften Aktivierung, Laufzeit und Verfügbarkeit der Rechenressource sein.
- Intensität: Sie bezeichnet die Größe der Störung. Diese richtet sich nach einer prozentualen Abweichung vom Sollwert. Bei der Aktivierung wird die Störung anhand der prozentualen Abweichung vom erwarteten Aktivierungszeitpunkt gemessen (in Jitter-behafteten Systemen kann auch eine verzögerte Aktivierung zu Timing Errors führen).

Bei der Laufzeit wird die Störung anhand der prozentualen Abweichung von der oberen und unteren Laufzeitgrenze der Task-Sektionen gemessen. Bei der Rechenressource bestimmt die Störung die Verringerung der freien Rechenzeit als Funktion eines Zeitintervalls.

- Dauer: Sie bezeichnet das zeitliche Auftreten der Störung. Man unterscheidet zwischen persistenten und transienten Störungen. Persistente Störungen sind dauerhaft vorhanden und daher besonders kritisch. Transiente Störungen sind über die Häufigkeit ihres Auftretens definiert. Störungen können bei einer begrenzten Anzahl zusammenhängend aufeinanderfolgender Instanzen einer Task auftreten, aber auch unter Auslassung von Task-Instanzen. (es)

**@ ARTIKEL-PDF ZUM DOWNLOAD**

Die Fortsetzung dieser Abhandlung lesen Sie unter
**www.hanser-automotive.de/Multicore**
Darin behandeln die Autoren Timing-Errors und betrachten die Dependability des Scheduling-Sub-Systems.
Eine simulationsbasierte Fallstudie bewertet die Robustheit von Multicore-Scheduling-Algorithmen.
Außerdem zeigen sie ein Absicherungsverfahren für persistente Störungen – den Online-Reliability-Observer.

*Literaturverzeichnis*

*[1] Partitionierungs-Scheduling von Automotive Restricted Tasksystemen auf Multiprozessorplattformen. M. Deubzer, U. Margull, J. Mottok, M. Niemetz, and G. Wirrer. Sindelfingen, Dezember 2009. Bd. Embedded Software En-gineering Congress.*

*[2] Funk, S. EDF Scheduling on Heterogeneous Multiprocessors. Dissertation at Department of Computer Science: University of North Caroline at Chapel Hill, 2004.*

*[3] S. Krämer, J. Mottok, H. Meier. OSEK-basierende Imple-*

*mentierung des LLREF-Scheduling-Algorithmus für eine Dual-Core-Architektur. Hanser Automotive, 2010, 1-2.*
*[4] Partly Proportionate Fair Multiprocessor Scheduling of Heterogeneous Task Systems. M. Deubzer, U. Margull, J. Mottok, M. Niemetz, and G. Wirrer. Toulouse, Mai 2010. Bd. Embedded Real Time Software and Systems Conference.*
*[5] Efficient Scheduling of Reliable Automotive Multicore Systems with PD$^2$ by Weakening ERfair Task System Requirements. M. Deubzer, J. Mottok, U. Margull, and M. Niemetz. Stuttgart, Juli 2010. Automotive Safety & Security 2010. S. 53-67.*
*[6] Effizientes Multicore-Scheduling in Eingebetteten Systemen – Teil 2: Ein simulationsbasierter Ansatz zum Vergleich von Scheduling-Algorithmen. M. Deubzer, F. Schiller, J. Mottok, M. Niemetz, and U. Margull. 2010, Bd. atp – Automatisierungstechnische Praxis.*
*[7] A categorization of real-time multiprocessor scheduling problems and algorithms. Handbook on Scheduling Algorithms, Methods and Models. Carpenter, J. and Funk, S. and Holman, P. and Srinivasan, A. and Anderson, James H and Baruah, Sanjoy K., 2004.*
*[8] Validating timing constraints in multiprocessor and distributed real-time systems. Ha, Rhan and Liu, J.W.S. 1994, Bd. International Conference on Distributed Computing Systems.*

**Prof. Dr. Jürgen Mottok** lehrt Informatik an der Hochschule Regensburg. Seine Lehrgebiete sind Software Engineering, Programmiersprachen, Betriebssysteme und Functional Safety. Er leitet das Laboratory for Safe and Secure Systems (LaS$^3$, www.las3.de).

**Michael Deubzer** arbeitet als wissenschaftlicher Mitarbeiter am Software Engineering Laboratory for Safe and Secures Systems (LaS$^3$). Er ist Doktorand in einem kooperativen Promotionsverfahren zwischen der Hochschule Regensburg und der TU München.

@ *Hochschule Regensburg*
*www.hs-regensburg.de*

**Andreas Bärwald** ist Product Line Manager Funktionale Sicherheit und Fachzertifizierer bei der TÜV Süd Automotive GmbH in Garching. Er ist weltweit fachlich für Funktionale Sicherheit innerhalb der Division Automotive des TÜV-Süd-Konzerns verantwortlich.

@ *TÜV Süd*
*www.tuev-sued.de*