OSEK/VDX

Markus Mayer

27th November 2002

serrarris@gmx.net

05.12.2002

1 Zusammenfassung

Der von der Automobilindustrie geschaffene Standard OSEK/VDX spezifiziert ein Echtzeitbetriebssystem für einen Prozessor, daß im Gegensatz zu den General Purpose Betriebssystemen keine dynamische Speicherverwaltung oder dynamisches Prozessmanagement besitzt. So ist zur Kompilierzeit schon die Anzahl der Prozesse bekannt, und welche Systemressourcen diese benötigen. Es ist aber z.B möglich mehrere Prozesse parallel laufen zu lassen, das heißt, das Betriebssystem ist multitaskingfähig. Dies und andere Eigenschaften von OSEK-OS werden im folgenden erläutert.

2 Allgemeines

2.1 Entstehung

Urspünglich verfolgten sowohl Autohersteller und Zulieferer in Deutschland als auch in Frankreich unabhängig voneinander das Ziel, ein modulares und statisches Echtzeitsystem für elektronische Systeme in Kraftfahrzeugen zu schaffen. Dies waren u.A. auf französischer Seite Renault und PSA (Peugot, Citroen), die ihr Projekt VDX ("Vehicle Distributed Executive") tauften und auf deutschem Gebiet z.B. Bosch, BMW, Daimler Benz, Opel und Siemens, welche ihre Initiative OSEK ("Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug") betitelten. 1994 schlossen sich dann diese beiden Gruppen zusammen, um schließlich 1995 die erste OSEK/VDX Spezifikation vorzustellen.

2.2 Ziele

Grundsätzlich sollte durch den Standard eine Senkung der Kosten für die Entwicklung und Weiterentwicklung von Software für ECU's (Electronic Control Units) erreicht werden. Dies wurde dann auch durch eine einheitliche Entwicklungsumgebung erreicht, die den beteiligten Firmen allgemeingültige Abläufe und Methoden zur Verfügung stellt. Mit Hilfe der Standardisierung von Schnittstellen sollte eine hohe Portabilität der von unterschiedlichen Herstellern gelieferte Software gewährleistet werden, was das Testen von dieser und der Hardware erheblich erleichtert. Außerdem sollte OSEK/VDX eine effizient designte Architektur, bei der alle Funktionen konfigurier- und skalierbar sind, aufweisen, um eine optimale Anpassung auf die jeweilige Anwendung zu bieten.

3 Struktur

Der Standard definiert drei Komponenten als eigenständige Bestandteile von OSEK/VDX:

1. OSEK-OS: Ein multitaskingfähiges Echtzeitbetriebssystem für einen Prozessor
2. OSEK-COM: Ein Kommunikationssystem
3. OSEK-NM: Netzwerk Management

Diese Standarts sind unabhängig voneinander und müssen nicht gemeinsam verwendet werden. Durch die Trennung der Komponenten ist es auch möglich, sie einzeln zu verbessern und zu erweitern. Daß davon Gebrauch gemacht wird, zeigen allein schon die Versionsnummern: OSEK-COM hat im Moment die Versionsnummer 3.0, OSEK-NM 2.5 und OSEK-OS 2.2. Im Folgenden wird nur OSEK-OS weiter betrachtet.

4 OSEK-OS

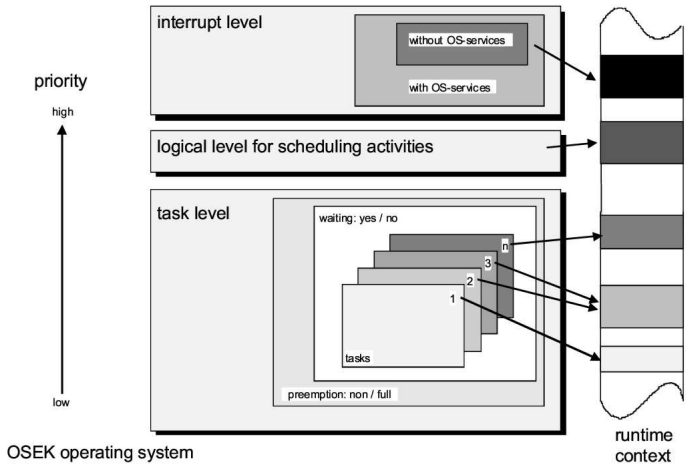
4.1 Architektur

OSEK-OS kann, im Gegensatz zu anderen Betriebssystemen, während der Laufzeit nicht mehr rekonfiguriert werden. Es wird bei der Systemherstellung mit der durch den Benutzer festgelegten Konfiguration generiert. Hier kann man z.B. verschiedene Eigenschaften des Systems bestimmen, welche in sogenannten "Conformance Classes" (Konformitätsklassen) zusammengefasst sind. Diese sind bedingt durch die unterschiedlichen Leistungsmerkmale eines Systems (Prozessor, Speicher) und die unterschiedlichen Anforderungen der Softwareanwendungen. Sie sind u.A. eingeführt worden um dem Benutzer einen leichteren Zugang zum System zu ermöglichen.

Dem Benutzer steht eine bestimmte Menge an Schnittstellen zur Verfügung, die von folgenden zwei Prozesstypen genutzt werden:

1. Interruptroutinen, die vom Betriebssystem verwaltet werden
2. Tasks (Standard und erweiterte Tasks)

Diese zwei Typen bilden jeweils eine eine Prioritätenklasse, dazu kommt noch eine Dritte, die der Scheduler belegt. Ihre Rangordnung (und damit auch die Reihenfolge bei der Prozessorbelegung) kann man aus folgender Grafik ersehen:



4.2 Tasks

4.2.1 Konzept

Ein wesentliches Betriebsmittel von OSEK-OS sind die Tasks. Von diesen gibt es zwei verschiedene Arten: Die Standard- und die erweiterten Tasks. Standardtasks geben den Prozessor nur frei, wenn sie terminieren oder vom Scheduler oder einem Interrupt dazu gezwungen werden. Erweiterte Tasks dagegen haben die Möglichkeit in einen "Blockiert"-Zustand ("Waiting") zu gehen, in dem sie den Prozessor wieder freigeben, aber ihren aktuellen Zustand beibehalten, und aus dem sie durch sogenannte "Events" (Ereignisse) wieder befreit werden können. Über diese hat man auch die Möglichkeit, das Verhalten der erweiterten Tasks zu beeinflussen. Auf dem Prozessor kann jeweils nur ein Task gleichzeitig laufen. Dehalb exisieren für die Tasks mehrere Zustände: Sie können "Bereit" bzw. "Ready" sein, d.h. sie warten nur noch darauf, das der Scheduler sie dem Prozessor zuteilt, oder sie werden gerade ausgeführt ("Laufend" bzw. "Running"), oder sie verhalten sich passiv und warten darauf, aktiviert zu werden ("Suspendiert"). Dieser letzte Zustand resultiert aus der Tatsache, das alle Tasks, die verwendet werden, schon zur Systemgenerierung bekannt sind. Bei den erweiternen Tasks kommt noch der oben schon erwähnte "Blockiert"-Zustand hinzu.

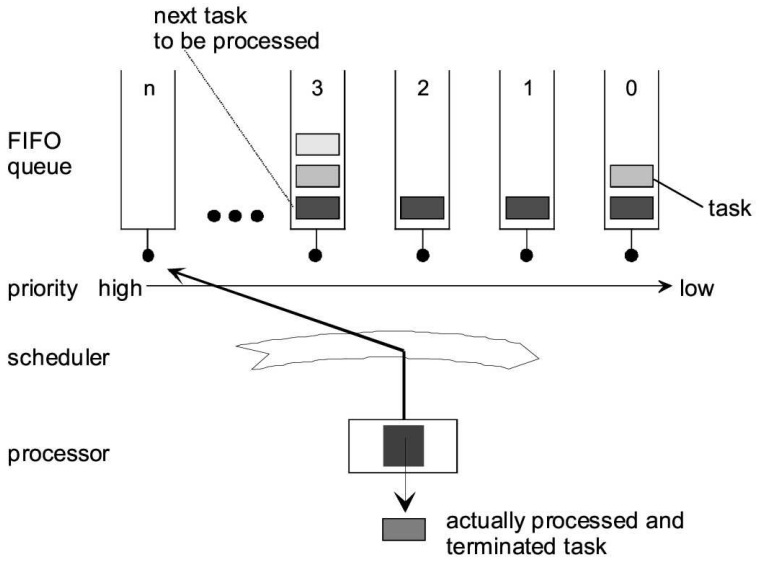
Um die Komplexität des Betriebssystems möglichst gering zu halten, wurden im Bereich der Tasks mehrere Maßnahmen getroffen. So können erweiterte Tasks z.B. nicht direkt vom Start aus in den Warte-Zustand gehen, sondern muessen erst in den "Running" Zustand gelangen, außerdem können sie im "Blockiert"-Zustand nicht terminieren. Ein Task kann sich überhaupt, im Gegensatz zu anderen Betriebssystemem, nur selbst beenden.

Der Vorteil von Standardtasks gegenüber der komplexeren erweiterten Tasks lieg darin, das diese nur wenig Systemressourcen belegen - aber dafür stehen als Synchronisationspunkte (d.h. Stellen, an denen sich der Task den Ausführungszeiten anderer angleichen kann) nur der Aufruf und das Ende des Tasks zur Verfügung. Anwendungen mit mehreren Synchronisationpunkten müssten somit auf mehrere Standardtasks verteilt werden. Ein erweiterter Task kann dies alleine bewerkstelligen, egal wie viele Synchronisationsanforderungen an ihn gestellt werden. Wann immer Information fehlt, wechselt er in den "Blockiert"-Zustand. Diesen Zustand verlässt der Task, wenn ein entsprechendes Ereignis ihm mitteilt, das die Resource, auf die er wartet, jetzt zur Verfügung steht.

Tasks werden entweder durch einen System-Call gestartet (z.B. von einem anderen Task), es gibt aber auch dir Möglichkeit, für einen Task einen Nachfolger festzusetzen, so das eine Ausführungskette definiert werden kann. Bei Standardtasks ist auch ein Mehrfachaufruf bis zu einer bestimmten Anzahl möglich, diese Tasks exisieren dann parallel.

4.2.2 Scheduling

Der Scheduler ruft die Tasks in der Reihenfolge ihrer Priorität auf. Aus Effizienzgründen sind die Prioritäten eines Tasks statisch, d.h. sie können zur Systemlaufzeit nicht mehr verändert werden. Es besteht in Ausnahmefällen aber die Möglichkeit, daß das Betriebssystem einen Task so behandelt als hätte er eine höhere Priorität (dazu später). In Konformitätsklassen, in denen mehrere Tasks mit gleicher Priorität erlaubt sind, werden diese wie in der folgenden Grafik zu sehen ist behandelt:



Zu Erläuterung: Tasks mit gleicher Priorität werden in der Reihenfolge ihres Aufrufs abgearbeitet, und erst wenn alle Tasks der höchsten Priorität abgearbeitet sind, wird mit der Nächstniedrigeren forgefahren.

Es gibt zwei Möglichkeiten, wie der Scheduler arbeiten kann:

1. Unterbrechendes Scheduling: Ein Task, der im Moment läuft kann zu jeder Zeit in den "Ready" zurückgesetzt werden, damit ein Task höherer Ordnung den Prozessor belegen kann. Dieses Verfahren bewirkt, das die Latenzzeit des Systems für Tasks höherer Ordnung unabhängig von der Laufzeit niedrigprioritärer Tasks ist. Allerdings wird dadurch, das sich u.U. mehrere Tasks "schlafen" legen müssen der Speicher eines Systems sehr belastet.

2. Nicht unterbrechendes Scheduling: Ein Task, der aufgerufen wurde belegt den Prozessor solange, bis er ihn selbst wieder freigibt. Das kann dazu führen das Tasks niedriger Priorität den Pozessor für Tasks höherer Priorität lange blockieren.

Diese beiden Arbeitsweisen eines Schedulers können auch gemischt werden, so daß Tasks selbst bestimmen können, auf welche Art sie behandelt werden möchten. Es ist auch möglich Tasks in Gruppen zusammenzufassen, wobei dann gilt, das Tasks einer Gruppe nicht unterbrochen werden, solange nicht ein Task aufgerufen wird der höhere Priorität besitzt als der Task der Gruppe mit der größten Priorität.

4.3 Synchronisation

4.3.1 Events

Events stehen wie schon erwähnt nur den erweiterten Tasks zu Verfügung. Sie dienen als Arbeitsmittel zu Synchronisation, und können erweiterte Tasks vom "Blockiert" in den "Ready" Zustand bringen.

Ein Event ist ein Objekt, das vom Betriebssystem verwaltet wird, und jeweils einem Task (dem "Besitzer") zugewiesen ist. Wenn dieser Task gestartet wird, werden alle ihm zugewiesenen Events gelöscht. Die Bedeutung von Events (ob ein Event z.B. einen ausgelaufenen Timer oder die Verfügbarkeit einer Resource anzeigt) ist von der jeweiligen Anwendung der der Task angehört vorgegeben.

Ein Event kann von jedem Task, auch von Standardtasks, und von bestimmten Interruptroutinen gesetzt werden, aber nur dem Besitzertask ist es erlaubt ein Event zu löschen oder auf ein Event zu warten. Wenn mindestens ein Event eines Tasks aktiviert wird, während sich der Task im "Warte"-Zustand befindet, dann wird dieser Task auf "Ready" gesetzt.

4.3.2 Resourcen

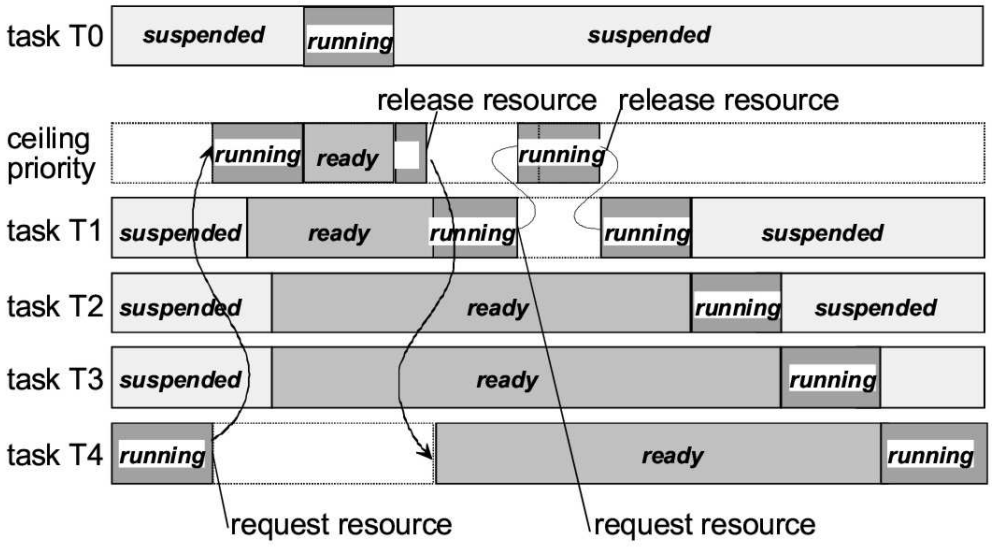
Die Resourcenverwaltung legt die Zugriffsrechte von Tasks (oder u.U. auch Interruptroutinen) unterschiedlicher Priorität auf gemeinsame Resourcen wie z.B. Verwaltungseinheiten oder Speicher fest. Was stellt diese Verwaltung sicher?

- Mehrere Tasks können nicht gleichzeitig auf dieselbe Resource zugreifen.
- Prioritätenumdrehung und Deadlocks können nicht durch Zugriff auf Ressourcen auftreten.
- Der Versuch, auf eine Resource zuzugreifen führt nie dazu, das der Task in einen "Warte"-Zustand übergehen muss

Damit dies gelten kann, darf, wenn eine Resource belegt ist, kein Task aktiviert oder der Scheduler neu aufgerufen werden. Die Vermeidung von Prioritätenumdrehung und Deadlock werden über das Sogennante OSEK Priority Ceiling Protokoll erreicht, welches wie folgt funktioniert:

- Jeder Resource wird bei der Systemgeneration eine Priorität zugewiesen. Diese muß größer (oder gleich) sein als alle Prioritäten von Tasks, die auf diese Resource (oder eine mit dieser verknüpften Resource) zugreifen, und kleiner als alle, die nicht auf diese Ressource zugreifen und eine höhere Priorität haben als Tasks, die auf die Resource zugreifen.
- Wenn ein Task auf eine Resource zugreift, wird seine Priorität auf die Priorität der Resource angehoben.
- Wenn er diese wieder freigibt, bekommt er wieder seine ursprüngliche Priorität zugewiesen.

Somit können Tasks, die vielleicht auf dieselbe Resource zugreifen würden nicht in den "Running"-Zustand übergehen, weil sie eine niedrigere Priorität als der im Moment laufende, die Resource belegende, Task haben. Ein Beispiel dazu (es wird unterbrechendes Scheduling verwendet):



Task T0 hat die hoechste, Task T4 die niedrigste Priorität, T1 und T4 wollen dieselbe Resource belegen, und T4 wird als erstes aufgerufen. Die Priorität von T4 wird dadurch erhöht, und sobald T4 die Resource wieder freigegeben hat ist seine Priorität so klein, das T0 den Prozessor zugeteilt bekommt.

4.4 Interupt Management

Interruptroutinen lassen sich in zwei Kategorien einteilen:

1. Die Interruptroutine benutzt keine Systemdienste. Dann hat der Interrupt keine Auswirkung auf das Taskmanagement, weil nach Ende der Routine der Prozessor seine Arbeit genau an der Stelle wieder aufnimmt, an der der Interrupt die normale Abarbeitung der Tasks gestört hat.

2. Wenn die Interruptroutine Systemdienste nutzt stellt ihr das OSEK System eine Laufzeitumgebung zur Verfügung, die schon zur Systemerzeugung dem Interrupt zugewiesen wurde.

Während der Ausführung einer Interruptroutine wird kein Rescheduling vorgenommen. Bei Routinen vom Typ 2 wird, wenn diese terminiert ist, der Scheduler aufgerufen, falls ein Task unterbrochen wurde und kein anderer Interrupt aktiv ist. Das Scheduling von Interupts wird von der Hardware übernommen (im Gegensatz zu Tasks). Interrupts können sowohl unterbrechbare als auch nicht unterbrechbare Task vom Prozessor verdrängen! Wenn ein Task von einer Interruptroutine aktiviert wurde, wird nach Termination der Interruptroutine dieser dem Scheduler übergeben.

References

[1] www.osek-vdx.org

[1] www.3soft.de/e/proosek

[2] www.kfzelectronik.de/b_osek.htm