Real-Time Operating Systems – Ein Überblick

Stefan Tittel
Universität Dortmund

Proseminar: Werkzeuge und Techniken zur Spezifikation,
Simulation und Implementierung von eingebetteten Systemen, 2004

1 Einführung

1.1 Operating Systems

1.1.1 Definition: Operating System

Das *Operating System* ist das Programm, welches beim Start eines Rechners geladen wird. Es dient als eine Schnittstelle zwischen den Rechnerkomponenten wie Hardware und BIOS und den Anwendungen, die auf dem Rechner laufen. Es bietet außerdem grundlegende Funktionen für die Verwaltung und Pflege des Operating System und Dateisystems.

1.1.2 Aufgaben eines Operating System

Verwaltet:

- Rechenleistung (Scheduling)
- Speicher
- Dateisystem
- Geräte

Stellt bereit:

- Hardwareabstraktion gegenüber Applikationen
- Benutzerschnittstelle gegenüber Anwendern

Berücksichtigt dabei:

- Effizienz
- Sicherheit

1.1.3 Eigenschaften eines Operating System

- Ein Operating System ist im Allgemeinen sehr komplex.
- Ein Operating System versucht im Allgemeinen unterschiedliche, teils gegensätzliche Anforderungen gut zu erfüllen (z. B. Effizienz trotz Sicherheit).
- Das Verhalten des Operating System ist dadurch nicht mit vertretbarem Aufwand determinierbar.

1.2 Notwendigkeit eines Real-Time Operating System

1.2.1 Motivation

Oftmals muss ein System auf ein Ereignis in gewissen Grenzen temporal determinierbar reagieren. Ein herkömmliches Operating System leistet dies in der Regel nicht oder in nicht ausreichendem Maße.

1.2.2 Definition: Real-Time System

Ein *Real-Time System* ist ein System, dessen Korrektheit nicht nur korrekte Berechnungen verlangt, sondern darüber hinaus zeitliche Vorgaben über das Eintreffen des Ergebnisses macht. Ein Real-Time System reagiert auf zeitlich vorhersehbare Weise auf das Eintreten externer Stimuli.

1.2.3 Definition: Real-Time Operating System

Ein *Real-Time Operating System* ist ein Operating System mit den notwendigen Eigenschaften zum Betrieb eines Real-Time System.

1.3 Klassifizierung von Real-Time Systems

1.3.1 Definition: Soft Real-Time System

Stellt in einem System das gelegentliche Verpassen von Deadlines eine akzeptable Qualitätsminderung dar, so handelt es sich um ein *Soft Real-Time System.*

1.3.2 Definition: Firm Real-Time System

Stellt in einem System das Verpassen von Deadlines eine nicht akzeptable Qualitätsminderung dar, so handelt es sich um ein *Firm Real-Time System.*

1.3.3 Definition: Hard Real-Time System

Wenn das Verpassen einer Deadline katastrophale Folgen nach sich zieht, handelt es sich um ein *Hard Real-Time System.*

2 Scheduling

2.1 Grundlegende Definitionen

2.1.1 Definition: Worst-Case Execution Time

Die *Worst-Case Execution Time (WCET)* ist die obere Grenze der Ausführungszeit eines Task.

2.1.2 Definition: Periodizität von Tasks

Ein Task, der alle p Zeiteinheiten ausgeführt wird, heißt *periodisch*; p ist dabei seine *Periode*. Tasks, die nicht periodisch sind, heißen *nicht-periodisch*.

Nicht-periodische Tasks, die zu nicht vorhersehbaren Zeitpunkten eintreffen, heißen *sporadisch*

2.1.3 Definition: Dynamisches und statisches Scheduling

Werden die Scheduling-Entscheidungen während der Laufzeit getroffen, so handelt es sich um *dynamisches Scheduling*.

Werden die Scheduling-Entscheidungen bereits zur Zeit der Entwicklung festgelegt, handelt es sich um *statisches Scheduling*.

Systeme mit Zeitgeber, die Tasks ausschließlich nach Start- und Stoppzeiten schedulen, heißen *entirely time triggered (TT)*.

2.1.4 Definition: Kooperatives und präemptives Scheduling

Nimmt der Scheduler einen Kontextwechsel nur dann vor, wenn der laufende Task dies explizit anbietet oder beendet ist, handelt es sich um *kooperatives Scheduling*.

Nimmt der Scheduler einen Kontextwechsel vor, ohne dass die Bedingungen für kooperatives Scheduling erfüllt sind, handelt es sich um *präemptives Scheduling*.

2.1.5 Weitere Merkmale

- Ein- oder Mehrprozessor-Scheduling
- Online- oder Offline-Scheduling
- Scheduling mit abhängigen oder unabhängigen Tasks

2.1.6 Weitere Definitionen

Die Differenz zwischen der Ausführungszeit eines Tasks und dem Deadline-Intervall heißt *Laxity*.

Die Differenz zwischen der Fertigstellungszeit und der Deadline – maximiert über alle Tasks – heißt *Maximum Lateness*. Wenn kein Task seine Deadline überschreitet, ist die Maximum Lateness negativ.

Ein Menge von Tasks heißt *schedulable*, wenn ein Schedule der Form existiert, dass jeder Task seine Deadline erfüllt.

2.2 Algorithmen

2.2.1 Earliest Due Date (EDD)

Voraussetzungen nicht-periodische, voneinander unabhängige, gleichzeitig eintreffende Tasks; Einprozessorsystem

Merkmale kooperativ; kann statisch implementiert werden, wenn Ausführungszeiten aller Tasks bekannt

Vorgehensweise

1. Sortiere alle Tasks nach Deadline aufsteigend.
2. Führe die Tasks gemäß dieser Sortierreihenfolge aus.

2.2.2 Earliest Deadline First (EDF)

Voraussetzungen nicht-periodische, voneinander unabhängige Tasks; Einprozessorsystem

Merkmale präemptiv; dynamisch

Vorgehensweise

1. Führe immer den Task mit der am nächsten liegenden Deadline aus (ähnlich *EDD*).
2. Bei Neuankunft eines Tasks mit früherer Deadline, wechsele zu diesem.

2.2.3 Least Laxity (LL)

Voraussetzungen nicht-periodische, voneinander unabhängige Tasks, bei denen die Länge ihrer Ausführungszeit im Voraus bekannt ist; Einprozessorsystem

Merkmale präemptiv; dynamisch

Vorgehensweise Führe immer den Task mit der geringsten *Laxity* aus.

2.2.4 Latest Deadline First (LDF)

Voraussetzungen nicht-periodische, voneinander abhängige, gleichzeitig eintreffende Tasks, von denen die Länge ihrer Ausführungszeit im Voraus bekannt ist; Abhängigkeiten liegen als Halb- oder vollständige Ordnung vor; Einprozessorsystem

Merkmale kooperativ; dynamisch

Vorgehensweise

1. Liegen die Abhängigkeiten als Halbordnung vor, erzeuge mittels topologischer Sortierung eine vollständige Ordnung.
2. Führe die Tasks in der Reihenfolge der vollständigen Ordnung aus.

2.2.5 Rate Monotonic Scheduling (RM)

Voraussetzungen

- Periodische, voneinander unabhängige Tasks
- Deadline und Periodendauer bei jedem einzelnen Task gleich lang
- Länge alle Ausführungszeiten im Voraus bekannt
- Es gilt $\sum_{i=1}^{n} \frac{c_i}{p_i} \leq n(2^{\frac{1}{n}} - 1)$, wobei $i \in \{\text{Tasks}\}$, c_i Ausführungszeit, d_i Dead- line-Intervall, p_i Periodendauer und n Anzahl der Tasks.

Vorgehensweise Vergib Prioritäten nach Periodenlänge: Je kürzer p_i desto höher die Priorität.

3 Embedded Linux

3.1 Bestandsaufnahme

3.1.1 Linux als Betriebssystem für Soft Real-Time Systems

- Linux unterstützt die Vergabe von statischen Prioritäten.
- Kernelpatches mit alternativen Schedulern sind verfügbar.
- Dank offener Quellen ist die Entwicklung eines Schedulers mit dynamischen Prioritäten denkbar.

3.1.2 Linux als Betriebssystem für Hard Real-Time Systems

- Mangelnde Scheduling-Fähigkeiten
- Ungeeignetes Interrupt-Handling
- Optimierung auf Effizienz, nicht auf Einhaltung von Deadlines

3.2 Vom Non-Real-Time zum Real-Time Operating System

3.2.1 Implementierung von Hard Real-Time

- Hardware- und Interruptverwaltung durch Real-Time Kernel
- Linux-Kernel läuft als Idle-Task, wenn Real-Time Kernel unbeschäftigt.
- System erhält zum eigentlichen Kernel- und User-Space einen Realtime-Space.
- Interrupt-Anforderungen des Linux-Kernel werden vom Real-Time Kernel abgefangen.

3.2.2 Real-Time Space

Real-Time Tasks im Real-Time Space sind völlig autonom vom Rest des Systems, d. h.

- kein Schutz durch Linux-Kernel
- kein Einsatz konventioneller Debugger
- keine Interprozess-Kommunikation ohne weitere Maßnahmen, wie z. B.
  - FIFOs
  - Shared Memory
  - Mailboxes
  - Messages

Literatur

[1] Peter Marwedel. *Embedded System Design*. Kluwer Academic Publishers, a. a. O., 2003

[2] Raquel S. Whittlesey-Harris [online]. *Real-Time Operating Systems*. Internet: `<http://tinyurl.com/3dgb6>`, 2001 [angeführt 2004-04-10]

[3] Robert Schwebel. *Embedded Linux – Handbuch für Entwickler*. mitp-Verlag, Bonn, 2001

[4] Rüdiger Brause. *Betriebssysteme – Grundlagen und Konzepte*, 2. Aufl., Springer-Verlag, Berlin/Heidelberg, 2001

[5] Mathai Joseph, Alan Burns, Andy Wellings, Krithi Ramamritham, Jozef Hooman, Steve Schneider, Zhiming Lui, Henk Schepers [online]. *Real-time Systems Specification, Verification and Analysis*. Internet: `<http://www.tcs.com/techbytes/htdocs/RTSbook.zip>`, 2001 [angeführt 2004-04-10].