3 Echtzeit, Echtzeitsysteme, Echtzeitbetriebssysteme

Kapitelübersicht

Bisher haben wir für die Reaktionszeit eines eingebetteten Systems keine zeitlichen Vorgaben (engl. deadlines) gemacht. Dies gilt nicht für alle eingebetteten Systeme: Einige davon fallen in die Kategorie der Echtzeitsysteme. In diesem Kapitel werden wir Charakteristika und Herausforderungen von Echtzeit- sowie Echtzeitbetriebssystemen kennenlernen.

DIN 44300

Die *DIN 44300* (DIN = Deutsche Industrienorm) beschreibt den Begriff Echtzeit wie folgt: Unter Echtzeit versteht man den Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind, derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zeitlich zufälligen Verteilung oder zu vorherbestimmten Zeitpunkten anfallen.

3.1 Echtzeitsysteme

Definition (Echtzeit)

Das Oxford Dictionary of Computing definiert Echtzeitsysteme wie folgt (freie Übersetzung):

Ein Echtzeitsystem ist ein System, bei dem der Zeitpunkt, zu dem Ausgaben erzeugt werden, bedeutend ist. Das liegt für gewöhnlich daran, dass die Eingabe mit einigen Änderungen der physikalischen Welt korrespondiert und die Ausgabe sich auf diese Änderungen beziehen muss. Die Verzögerung zwischen der Zeit der Eingabe und der Zeit der Ausgabe muss ausreichend klein für eine akzeptable „Rechtzeitigkeit“ (engl. timeliness) sein.

Echtzeitsysteme sind also Systeme, die korrekte Reaktionen innerhalb einer definierten Zeitspanne produzieren müssen. Falls die

Reaktionen diese Zeitlimits überschreiten, führt dies zu Leistungseinbußen und/oder Fehlfunktionen.

Die korrekte Funktionsweise eines Echtzeitsystems ist demnach nicht nur von der logischen Korrektheit der Ergebnisse seiner Verarbeitungsschritte abhängig, sondern auch von dem *Zeitpunkt* an dem die Ergebnisse produziert wurden. Es handelt sich selbst dann um ein inkorrektes Verhalten, wenn das Ergebnis nicht zum richtigen Zeitpunkt vorliegt und nicht nur dann, wenn das Ergebnis selbst falsch ist. Ein Fehler in der Reaktion, oder mit anderen Worten ein falscher Reaktionszeitpunkt ist daher genauso falsch wie eine falsche Reaktion: Ein richtiges Ergebnis zur falschen Zeit ist ein Fehler.

Echtzeitsysteme können in harte und weiche Echtzeitsysteme unterteilt werden:

Harte Echtzeitsysteme

Bei einem *harten Echtzeitsystem* hat das Verpassen einer Deadline katastrophale Folgen für das System. Für solche Systeme ist es zwingend erforderlich, dass Reaktionen innerhalb einer vorgegebenen Deadline erfolgen. Mit anderen Worten könnte man also sagen, die Wahrscheinlichkeit, dass dies der Fall ist, muss 100% sein. Ein Beispiel hierfür ist das Flugsteuersystem (engl. flight control system) eines Flugzeugs. Unter einer *harten Echtzeitbedingung* versteht man eine zeitliche Bedingung, die vom System stets erfüllt werden muss, da auch nur gelegentliche Verletzungen erhebliche Folgen (z. B. die Schädigung der Umgebung) nach sich ziehen würde.

Weiche Echtzeitsysteme

In *weichen Echtzeitsystemen* ist die Einhaltung von Deadlines zwar wichtig, sie funktionieren jedoch weiterhin korrekt, wenn diese Deadlines verpasst werden. Unter einer *weichen Echtzeitbedingung* versteht man eine zeitliche Bedingung, bei der gelegentliche Verletzungen tolerierbar sind. Das typische Beispiel für weiche Echtzeitsysteme sind Multimediasysteme, die für diverse Systemleistungen eine gewisse Taktrate *nach Möglichkeit* erbringen sollten, z. B. die Darstellung von 25 Bildern pro Sekunde, wobei eine gelegentliche Verletzung dieser weichen Echtzeitbedingung eine meist tolerierbare Störung darstellt, nämlich beispielsweise ein leichtes Bildflackern.

Zeitschranken

Eine weitere Unterscheidung bei den Echtzeitbedingungen ergibt sich daraus, dass Zeitschranken die Dauer einer Zeitspanne theoretisch von oben, von unten oder auch von beiden Seiten beschränken können. In der Literatur wird oft nur den *oberen Zeitschranken*, welche die maximal erlaubte Zeitspanne festlegen, Beachtung geschenkt (Fränzle, 2002). *Untere Zeitschranken*, welche Anforderungen an die minimale Dauer einer Phase zum Ausdruck bringen oder auch *beidseitige Zeitschranken* werden hingegen selten

diskutiert. Dies liegt daran, dass obere Zeitschranken eine Anforderung an die Mindestgeschwindigkeit des eingebetteten Systems oder Teilen hiervon stellen, welche im Allgemeinen wesentlich weitergehende konstruktive Maßnahmen erfordert als eine zeitliche Beschränkung von unten, die im Prinzip durch Warten erfüllt werden kann. Wenn aber untere Zeitschranken in Kombination mit oberen Zeitschranken auftreten (beidseitige Zeitschranken) muss ihnen wieder besondere Beachtung geschenkt werden.

Beispiel: Zeitschranken

Ein Beispiel für obere sowie beidseitige Zeitschranken kann in der digitalen Zündelektronik einer Motorsteuerung gefunden werden (Fränzle, 2002). Obere Zeitschranken ergeben sich hier bei der Berechnung des Drehwinkels sowie der Impulszählung der Drehzahlberechnung. Beide Berechnungen müssen in der Lage sein, alle 143 Mikrosekunden einen Impuls entgegen zu nehmen. Auch die Klopferkennung und die Zündzeitpunktberechnung müssen jeweils ihre Berechnungen innerhalb dieser Zeitspanne erledigt haben. Eine weitere, enge beidseitige Zeitschranke von nur 12 Mikrosekunden Breite ergibt sich für die Zündimpulserzeugung aus der Notwendigkeit, den Zündwinkel auf ein halbes Grad genau einzustellen.

3.2 Ereignissteuerung versus Zeitsteuerung

Eingebettete Systeme können in ereignisgesteuerte (engl. event triggered) und zeitgesteuerte (engl. time triggered) Systeme unterschieden werden, wobei der größere Teil der ersten Kategorie zuzuordnen ist.

Ereignisgesteuerte Systeme

Ereignisgesteuerte Systeme werden durch Unterbrechungen gesteuert. Liegt an einem Sensor ein Ereignis (der Umgebung) vor, so wird eine Unterbrechung (engl. interrupt) ausgelöst. Dieses Vorgehen führt zu kurzen Reaktionszeiten, ist aber bei der Bearbeitung vieler gleichzeitiger Ereignisse (sogenannte event showers) anfällig. Darüber hinaus sind die zeitlichen Abläufe im Systeme schwer planbar und manchmal auch schwer nachvollziehbar.

Zeitgesteuerte Systeme

Bei *zeitgesteuerten Systemen* erfolgt keine Reaktion auf Eingabeereignisse, sondern Unterbrechungen werden lediglich durch periodische Zeitgeber ausgelöst. Sensoren werden vom Steuergerät aktiv selbst abgefragt. Dieses Verfahren wird als Polling bezeichnet. Hierbei ist es wichtig, passende Pollingintervalle zu wählen: Kurze Signale müssen ggf. gepuffert werden, um sie nicht zu verlieren. Der Vorteil zeitgesteuerter Systeme liegt darin, dass das zeitliches Verhalten sämtlicher Systemaktivitäten bereits vor der Laufzeit vollständig planbar ist. Dies ist gerade für den Einsatz in Echtzeit-

systemen ein erheblicher Vorteil, da a priori überprüft werden kann, ob Echtzeitanforderungen eingehalten werden können.

In eingebetteten Systemen ist die Zeitsteuerung schon seit mehreren Jahren ein etabliertes Konzept. Sie sorgt durch ihren zweistufigen Entwicklungsansatz dafür, dass die Integration von Komponenten verschiedener Zulieferer auf Anhieb klappt. Das Time-Triggered Protocol (TTP) (Kopetz et al., 2002) beispielsweise kommt in Fly-by-Wire Cockpits von Honeywell zum Einsatz und wird auch von Alcatel als Feldbusprotokoll in der Bahnhofssignalsteueranlage Elektra 2 eingesetzt. TTP, FlexRay Protocol und TTCAN gehören zurzeit zu den aktuellen zeitgesteuerten Technologien die auch ein eingebettetes Betriebssystem beherrschen sollte.

3.3 Echtzeitbetriebssysteme

Wie wir bereits diskutiert haben, werden an eingebettete Systeme oft Echtzeitanforderungen gestellt. Ein Echtzeitsystem ist ein System, bei dem der Zeitpunkt, zu dem Ausgaben erzeugt werden, bedeutend ist. Programme, die auf einer (fast) beliebigen Hardware ablaufen, die Grundfunktionen von Betriebssystemen erfüllen und Echtzeitverhalten aufweisen nennt man Echtzeitbetriebssysteme.

Echtzeitbetriebssysteme erfüllen dieselben Aufgaben wie ein normales Betriebssystem wie etwa die Verwaltung von Betriebsmitteln, Prozessen, Prozessor, Speicher und Peripherie. Sie werden überwiegend in komplexen eingebetteten Systemen benötigt, da verschiedenste Aufgaben gleichzeitig abgearbeitet werden müssen. In weniger komplexen eingebetteten Systemen werden sie selten integriert, da sie dort nur unnötig Ressourcen belegen würden und zu keiner Leistungssteigerung beitragen könnten. Man findet sie in zunehmenden Maße schon in fast allen Alltagsgegenständen wie beispielsweise Drucker, Automobil (Navigationssystem, Bordcomputer, ABS-Steuerung usw.), Handy, PDA, Waschmaschine, Kühlschrank, DVD-Spieler u.v.m. Dort werden sie manchmal auch als „Firmware“ bezeichnet.

Abschnittsübersicht

In diesem Abschnitt geben wir eine Einführung in das Gebiet der Echtzeitbetriebssysteme. Zu diesem Zweck wiederholen wir zunächst den allgemeinen Aufbau sowie Aufgaben von Betriebssystemen. Bei Echtzeitbetriebssystemen nimmt die Festlegung der zeitlichen Reihenfolge (das Scheduling) von Prozessen einen besonderen Stellenwert ein. Verschiedene Schedulingalgorithmen wollen wir deshalb ebenfalls in einem Abschnitt gesondert

betrachten. Schließlich werden einige in der Praxis weit verbreitete Echtzeitbetriebssysteme kurz vorgestellt.

Wie wir gesehen haben, ist der Begriff „Echtzeit“ nicht mit einer hohen Rechenleistung der unterliegenden Hardware gleichzusetzen. Unter Echtzeit ist vielmehr die *garantierte Bearbeitung* einer Aufgabe innerhalb einer *definierten Zeitspanne* zu verstehen.

3.3.1 Aufbau und Aufgaben von Betriebssystemen

Betriebssysteme bilden die Schnittstelle zwischen Hard- und Software. Unter einem Betriebssystem (engl. Operating System kurz: OS) versteht man Software, die zusammen mit den Hardwareeigenschaften des Computers die Grundlage zum Betrieb bildet und insbesondere die Abarbeitung von Programmen steuert und überwacht. In der DIN 44300 bzw. im Informatikduden wird ein Betriebssystem wie folgt definiert:

Definitionen (Betriebssystem)

Definition (Betriebssystem) nach DIN 44300:
Die Programme eines digitalen Rechensystems, die zusammen mit den Eigenschaften dieser Rechenanlage die Basis der möglichen Betriebsarten des digitalen Rechensystems bilden und die insbesondere die Abwicklung von Programmen steuern und überwachen.

Definition (Betriebssystem) nach dem Duden Informatik:
Zusammenfassende Bezeichnung für alle Programme, die die Ausführung der Benutzerprogramme, die Verteilung der Betriebsmittel auf die einzelnen Benutzerprogramme und die Aufrechterhaltung der Betriebsart steuern und überwachen.

Ebenfalls nach der DIN 44300 umfasst ein Betriebssystem die Programme eines digitalen Rechensystems, die zusammen mit den Eigenschaften dieser Rechenanlage die Basis der möglichen Betriebsarten des Rechensystems bilden und die insbesondere die Abwicklung von Programmen steuern und überwachen.

Aufgaben eines Betriebssystems

Ein Betriebssystem hat folgende *Aufgaben*. Es verwaltet:

- *Hardware-Betriebsmittel* (Prozessor, Speicher, Ein-Ausgabe-Geräte usw.) und Software-Betriebsmittel (Dateien, spezielle Programme)
- *Prozesse* (in Ausführung befindliche Programme)

Komponenten eines Betriebssystems

Das Betriebssystem besteht aus folgenden *Komponenten:*

- **Prozess-Verwaltung** (Kreieren und Terminieren von Applikations- und System-Prozessen, Suspendieren und Reaktivieren von Prozessen, Prozess-Synchronisation und Kommunikation, Deadlock-Behandlung)
- **Speicher-Verwaltung** ("Buchführung" über freie und belegte Hauptspeicherbereiche, Kommunikation mit den in der Hierarchie angrenzenden Speichermedien, Dynamische Allokation und Deallokation von Hauptspeicher, Speicherschutz und Zugriffskontrolle)
- **Prozessor-Verwaltung** (Dispatching, Scheduling, Unterbrechungsbehandlung)
- **Geräte-Verwaltung**
- **Datei-Verwaltung** (Datei-Konzepte, Datei-Attribute und Datei-Operationen, Zugriffs-Methoden, Verzeichnis-Strukturen und -Implementierungen, Allokations-Methoden)

Definition (Prozess)

Definition (Prozess):
Ein Prozess ist ein ablauffähiges Programm mit seinen dafür notwendigen betriebssystemseitigen Datenstrukturen wie zugeordneten Eigenschaften (z. B. Stack- und Programmzähler, Registerinhalte, Prozesszustand, sowie Eigenschaften der Speicher- und Dateiverwaltung).

Ein Prozess wird durch das Betriebssystem als eigenständige Instanz – in der Regel unabhängig und geschützt voneinander – ausgeführt. Multitaskingsysteme gestatten die nebenläufige Prozessausführung.

3.3.2 Betriebssystemarchitekturen

Universalbetriebssysteme wie etwa UNIX oder VMS besitzen meist einen relativ großen monolithischen Kern (engl. kernel), der einen umfangreichen Hauptspeicherbedarf besitzt. Neuere Betriebssysteme (z. B. Windows XP, Solaris oder Linux) hingegen benutzen entweder einen dynamischen Betriebssystemkern oder einen sehr kleinen Mikrokern (engl. microkernel). Dies birgt den Vorteil, die Ressource Hauptspeicher effizienter ausnutzen zu können. Folgende alternative Architekturen für Betriebssysteme sind möglich:

- **Monolithischer Betriebssystemkern:** Dieser Kern enthält die Treiber für alle Geräte, die evtl. an dem Rechner betrieben werden sollen und alle traditionellen Funktionen wie z. B. Prozessverwaltung, Prozessorverwaltung, Hauptspeicherverwaltung, Dateiverwaltung und Netzwerkdienste.
- **Dynamischer Betriebssystemkern:** Dieser Kern enthält nur die Funktionen, die häufig benutzt werden. Gerätetreiber oder zusätzliche Funktionen sind in separaten Modulen gekapselt, die je nach Bedarf in den Hauptspeicher geladen oder daraus entfernt werden.
- **Mikro-Betriebssystemkern:** Dieser Kern bietet nur fünf minimale Basisdienste an: Den Prozesskommunikationsmechanismus, eine einfache Speicherverwaltung, eine minimale Prozessverwaltung, ein einfaches Scheduling und eine einfache I/O-Funktionalität. Eine Aufgabe wird nur dann vom Kern erledigt, wenn die Funktionalität des Systems außerhalb des Kerns nicht gewährleistet ist. Mikro-Betriebssystemkerne sind unter gewissen Umständen weniger effizient, da häufig zwischen den Modi User und Supervisor umgeschaltet werden muss.

Anders als monolithische Betriebssystemkerne lagert ein Microkernel alle nicht unmittelbar zur Steuerung der Programmabläufe benötigten Funktionen in externe Subsysteme aus. Diese werden anschließend je nach Bedarf in den Kernel eingefügt. Auf diese Weise lässt sich das Betriebssystem für jeden Einsatzzweck anpassen und auch unter sehr restriktiven Rahmenbedingungen einsetzen.

3.3.3 Echtzeitfähige Betriebssysteme

Wie wir gesehen haben, ist der Begriff „Echtzeit“ nicht mit einer hohen Rechenleistung der unterliegenden Hardware gleichzusetzen. Unter Echtzeit ist vielmehr die *garantierte Bearbeitung* einer Aufgabe innerhalb einer *definierten Zeitspanne* zu verstehen.

Um die Echtzeitanforderungen zu erreichen, wurden früher für Echtzeitsysteme überwiegend genau auf die zu erfüllenden Aufgaben zugeschnittene und in Assembler geschriebene Programme verwendet. Diese Vorgehensweise bedingte allerdings hohe Entwicklungskosten, geringe Flexibilität und erforderte viel Spezialwissen seitens der Entwickler. Das Programm selbst musste viele typische Aufgaben, die heute ein Betriebssystem erledigt,

realisieren. Die Steigerung der Leistungsfähigkeit der Hardware bzw. der Anforderungen an die Systeme ermöglichte und erzwang immer mehr den Einsatz höherer Programmiersprachen bei der Softwareentwicklung auch für Echtzeitsysteme. Die Nutzung allgemein verfügbarer und damit kostengünstiger Komponenten für diese Systeme war eine weitere Motivation. Bei der Programmentwicklung greift man möglichst auf „Grundprogramme“ zurück, die sich durch Änderungen und Ergänzungen auf eine Anwendung zuschneiden lassen. Im Folgenden wollen wir Struktur und Eigenschaften von Programmen behandeln, die auf einer (fast) beliebigen Hardware ablaufen, die Grundfunktionen von Betriebssystemen erfüllen und Echtzeitverhalten aufweisen. Diese Programme nennt man Echtzeitbetriebssysteme.

Definition (Echtzeitbetrieb)

Definition (Echtzeitbetrieb) nach DIN 44300:
Echtzeitbetrieb ist ein Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind und zwar derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zeitlich zufälligen Verteilung oder zu vorbestimmten Zeitpunkten anfallen.

Echtzeitanforderungen

An eingebettete Systeme werden also oft Echtzeitanforderungen gestellt. Ein Echtzeitbetriebssystem führt zwischen der Hardware und den Applikationen eine Abstraktionsebene ein und ermöglicht es Softwareentwicklern auf diese Weise, von der Plattform unabhängige echtzeitfähige Applikationen zu programmieren.

Anforderungen an Echtzeitbetriebssysteme

Ein eingebettetes Betriebssystem sollte wenig Ressourcen verbrauchen und dabei stets zuverlässig und stabil laufen. Dabei dürfen Programmfehler weder das Betriebssystem noch andere Programme beeinflussen. Kurze Antwortzeiten sind erforderlich. Echtzeitsysteme und ihre Zeitanforderungen spiegeln die Größenordnung wider, unter denen Realzeitbetriebssysteme Verwendung finden. Hat man Zeitanforderungen im Minutenbereich, lässt sich ein System durchaus noch von Hand steuern, unterhalb dieser Marke (etwa eine Minute) reicht eine Automatisierung auf Basis von Mechanik aus. Zeitanforderungen im Sekunden-Bereich lassen sich mit einem Standardbetriebssystem erfüllen, im Millisekunden-Bereich benötigt man dagegen ein Realzeitbetriebssystem. Gilt es Anforderungen im Mikrosekunden-Bereich zu erfüllen, müssen die notwendigen Aktionen innerhalb einer Unterbrechungsroutine (engl. Interrupt Service Routing, kurz: ISR) durchgeführt werden. Unterhalb dieser Grenze hilft nur noch eine Realisierung in Hardware (Timmermann, 1997).

Zusammenfassend werden nach (Ber, 1996) an ein Echtzeitbetriebssystem folgende Anforderungen gestellt:

- Sie besitzen ein *deterministisches zeitliches Verhalten*, das besonders bei der zyklischen Einplanung von Prozessen notwendig ist,
- sie erlauben die Bearbeitung von vielen externen Ereignissen (mithilfe eines *Unterbrechungskonzepts*),
- die *Antwortzeiten* auf eine Unterbrechung sind definiert,
- sie gewährleisten eine *schnelle Reaktion,* d. h. einen geringen Overhead beim direkten Zugriff auf physikalische Hardware-Adressen bzw. geringer Overhead beim Kontextwechsel,
- eine *Multitaskingfähigke*it muss gegeben sein, die den Prozessor an andere laufwillige Prozesse vergibt, wenn die Prozessortätigkeit durch Interrupt, Timerablauf etc. unterbrochen wurde,
- lauffähige Programme (Tasks, Prozesse) müssen *prioritätsgesteuert* anlaufen,
- eine effiziente und *schnelle Interprozesskommunikation* muss möglich sein,
- durch ihre *Skalierbarkeit* erlauben sie ihren Einsatz in Systemen mit begrenzten Ressourcen,
- ihre *Last-Unabhängigkeit* muss garantiert werden,
- sie verwenden *Standardschnittstellen,*
- sie können auch mit *nicht echtzeitfähigen Systemen* kommunizieren,
- sie sind für unterschiedliche Prozessorarchitekturen verfügbar,
- es gibt *spezielle Entwicklungswerkzeuge,* die für Echtzeitanwendungen optimiert sind und
- ein guter und schneller Durchgriff auf den angeschlossenen *technischen Prozess* ist erforderlich.

Weist ein Betriebssystem diese Eigenschaften auf, so ist es prinzipiell für den Echtzeitbetrieb geeignet. Allerdings enthalten die oben genannten Eigenschaften noch keine tatsächlichen Zeitangaben. Antwortzeiten auf Unterbrechungen, Prozesswechselzeiten, Interprozess-Kommunikationszeiten oder gar die Angabe der Zeitäquidistanz bei zyklischer Einplanung müssen im Einzelfall genau überprüft werden.

Unterschiede zwischen Betriebssystemen und Echtzeitbetriebssystemen

Betriebssysteme in eingebetteten Systemen unterscheiden sich von Standardbetriebssystemen wie etwa Microsoft Windows, Unix, Linux usw. meist vor allem darin,

- dass sie deutlich kleiner (sowohl hinsichtlich des Codevolumens wie auch bezüglich des Funktionsumfangs) sind, um eine kosteneffiziente Integration in eingebettete Systeme zu erlauben, und
- dass sie spezielle Mechanismen zur Herbeiführung eines verlässlichen Zeitverhaltens der einzelnen Tasks enthalten (z. B. angepasste Prozessor- und Speicherverwaltung).

Aufgaben von Echtzeitbetriebssystemen

Echtzeitbetriebssysteme erfüllen grundsätzlich die gleichen *Aufgaben* wie „normale“ Betriebssysteme. Sie verwalten:

- Hardware-Betriebsmittel (Prozessor, Speicher, Schnittstellen, Ein-Ausgabe-Geräte usw.) und Software-Betriebsmittel (Dateien, spezielle Programme) und
- Prozesse (in Ausführung befindliche Programme)

und gestatten ferner den Aufruf von Systemprogrammen, die zu diesem Betriebssystem gehören oder zusätzlich bereit gestellt werden und stellen schließlich Programmierschnittstellen zur Verfügung, mit deren Hilfe eigene Programme die Funktionen des Betriebssystems nutzen können.

Zusätzliche Eigenschaften

Zusätzlich zu den allgemeinen Betriebssystemaufgaben müssen Echtzeitbetriebssysteme die Reaktionszeiten des Gesamtsystems, auf dem sie eingesetzt sind, gewährleisten. Diese werden von der verwendeten Hardware erheblich mitbestimmt.

Einschränkende Eigenschaften

Einschränkend verfügen Echtzeitbetriebssysteme nicht automatisch über eine Dateiverwaltung, da in vielen Anwendungen von Echtzeitbetriebssystemen keine mechanischen Laufwerke (Disketten, Festplatten) benötigt werden. Außerdem ist der Einsatz der Dateiverwaltung in Echtzeitbetriebssystemen problematisch, weil der Zugriff auf Datenträger zunächst selbst nichtdeterministisch ist, zum anderen aber auch durch die Interrupt Service Routine (ISR) zu Nichtdeterminismen im sonstigen System führen kann. Aus diesem Grunde ist bei Echtzeitbetriebssystemen die Speicherverwaltung ebenfalls oft einfacher aufgebaut als bei modernen Betriebssystemen. Hinzu kommt, dass viele Realzeitsysteme in Umgebungen eingesetzt werden, in denen aus Sicherheitsgründen bewegte Komponenten nicht zulässig sind. Damit scheiden ohnehin klassische Hintergrundspeichermedien (Festplatten, Diskettenlauf-

werke) aus; allerdings besteht hier technisch die Möglichkeit, sogenannte Flash-Speicher einzusetzen.

Nachstehende Tabelle gibt einen Überblick über die Unterschiede der beiden Betriebssystemarten:

Eingebettetes Betriebssystem	**Universalbetriebssystem**
Betriebssystem ist meistens in einem ROM abgelegt	Betriebssystem ist auf einer Festplatte abgelegt
Muss speichereffizient sein	Kann speichereffizient sein
Hat normalerweise keine GUI, da es meistens mit dem Embedded System interagiert	Hat überwiegend eine GUI, da es meistens mit dem Benutzer interagiert
Einarbeitungsaufwand meistens sehr hoch	Einarbeitungsaufwand eher gering
Wenige Applikationen bzw. Entwicklungswerkzeuge vorhanden	Reichliche Auswahl an Applikationen und Entwicklungswerkzeugen
Industrietauglich	Kaum industrietauglich
Geringe Lizenzkosten für den Einsatz in eingebetteten Systemen	Hohe Lizenzkosten für den Einsatz in eingebetteten Systemen
Betriebssystem ist deterministisch	Betriebssystem ist nichtdeterministisch

Tab. 3.1: Unterschiede zwischen eingebetteten und Standard-Betriebssystemen

Manche Universalbetriebssysteme bieten mittlerweile Systemerweiterungen an, die das System in die Lage versetzen, weiche Echtzeitanforderungen zu erfüllen. Bei diesen Erweiterungen handelt es sich um:

- Memory Locking
- Deaktivierung von Caches (führt allerdings gleichzeitig zu einem erheblichen Performanzverlust)
- Echtzeit Scheduling Strategien (POSIX Scheduling)

Beispiele: RTOS

Derzeit verfügbare, bekannte Echtzeitbetriebssysteme sind beispielsweise VxWorks (von Wind River), die Linux-nahen Echtzeitbetriebssysteme LynxOS (von Lynx-Works) und Linux-RT,

QNX (vom kanadischen Hersteller QNX Software Systems) und OSE (OSE Systems).

Die existierenden Echtzeitbetriebssysteme lassen sich im Wesentlichen in die zwei folgenden Kategorien unterteilen:

- Systeme, die auf optimierten Versionen von *konventionellen Betriebssystemen* aufsetzen wie z. B. Lynx und Linux-RT sowie
- Systeme, die von Grund auf *neu entwickelt* wurden, wie beispielsweise QNX, OSE und VxWorks.

3.3.4 Zeitgeber und Zugriffsebenen auf Zeit

Zur Bestimmung der Zeit gibt es in einem Echtzeitsystem sogenannte Realzeituhren bzw. Timer. Sie lassen sich prinzipiell auf zwei Arten realisieren: Sowohl in Hardware als auch in Software in Form von Vorwärts- und Rückwärtszählern. Bei Rückwärtszählern unterscheidet man repetitive Zähler von den Single Shot Zählern. Repetitive Zähler laden sich mit einem Zählwert selbständig nach Ablauf (wenn der Zählerstand 0 erreicht wird), während der Single Shot Zähler – wie bereits sein Name andeutet – explizit neu gestartet werden muss. Diese Zeitgeber erfüllen folgende Aufgaben:

Aufgaben von Zeitgebern

- **Zyklische Generierung von Unterbrechungen (Interrupts):** Ein Timer ist im System dafür verantwortlich, zyklisch (z. B. alle 10 ms) einen Interrupt zu generieren. In der zugehörigen Interrupt Service Routine (ISR) wird der Scheduler aufgerufen, es werden zeitabhängige Systemdienste (Weckaufrufe) bearbeitet und Softwaretimer realisiert. Damit hängt die Genauigkeit der zeitabhängigen Systemdienste von dieser Zeitbasis ab.
- **Zeitmessung:** Das Messen von Zeiten ist eine oft vorkommende Aufgabe in der Automatisierungstechnik. Geschwindigkeiten lassen sich beispielsweise über eine Differenzzeitmessung berechnen.
- **Watchdog (Zeitüberwachung):** Neben der Zeitmessung spielt die Zeitüberwachung eine sicherheitsrelevante Rolle in Echtzeitsystemen. Zum einen werden einfache Dienste (z. B. die Ausgabe von Daten an eine Prozessperipherie) zeitüberwacht, zum anderen aber auch das gesamte System (Watchdog). Dazu muss das System in regelmäßigen Abständen einen Rückwärtszähler zurücksetzen. Ist das System in einem undefinierten Zu-

stand und kommt nicht mehr dazu, den Zähler zurückzusetzen, zählt dieser bis 0 und bringt das System in den sicheren Zustand (löst beispielsweise an der CPU ein Reset aus).

- **Zeitsteuerung für Dienste:** Spezifische Aufgaben in einem System müssen in regelmäßigen Abständen durchgeführt werden. Zu diesen Aufgaben gehören Backups ebenso wie Aufgaben, die der Benutzer dem System überträgt (z. B. zu bestimmten Zeitpunkten Messwerte erfassen).

Dabei können nachstehende drei Zugriffsarten bzw. -ebenen auf die Zeit unterschieden werden:

Zugriffsebenen auf Zeit

(1) Hardware Level:
Auf Hardwareebene werden Timer in Form von Echtzeituhren (Absolutzeitgeber), Frequenzteilern, Rückwärtszählern und Watchdog Timern zur Verfügung gestellt.

Absolutzeitgeber halten die Absolutzeit (Tag, Monat, Jahr, Stunde, Minute, Sekunde, Millisekunde) vor. Diese Uhren sind batteriegepuffert, so dass auch nach Abschalten des Stroms die Uhrzeit weitergezählt wird. Absolutzeiten sind vor allem bei eingebetteten Systemen oftmals nicht notwendig. Insbesondere für verteilte Echtzeitsysteme jedoch ist eine genaue Absolutzeit erforderlich. Dabei ist es wichtig darauf zu achten, dass alle Teilsysteme die gleiche Zeitzone verwenden (zu einem Zeitpunkt ist die Uhrzeit unterschiedlich an den verschiedenen Orten der Erde). Um hier Probleme von vorneherein zu vermeiden, wird im Regelfall die Hardwareuhr auf die Zeitzone UTC (Universal Time) eingestellt. Die Betriebssysteme rechnen dann auf die lokale Zeit (mit Sommer-/Winterzeit usw.) um. Die eigentlichen Applikationen selbst jedoch holen sich die Zeit über Systemdienste (siehe unten) und sollten Zeiten nicht umrechnen müssen.

Über *Frequenzteiler* werden für die unterschiedlichen Hardwarekomponenten des Systems korrekte Frequenzen von einer Standardfrequenz abgeleitet (z. B. für die Baudrate einer seriellen Schnittstelle). Bei den Frequenzteilern handelt es sich um Dualzähler, an deren Zähleingang die Standardfrequenz gelegt ist. Über einen Multiplexer kann man einen der Ausgänge des Zählers auswählen. Dieser Ausgang liefert jetzt die entsprechend dem Ausgang zugeteilte Standardfrequenz.

Ein *Rückwärtszähler* dient zur Messung von Relativ-Zeiten (Zeitdifferenzen). Die Zähler werden mit einem Wert belegt und zählen dann mit einer Eingangsfrequenz (die über einen Frequenz-

teiler mit Multiplexer eingestellt werden kann) auf 0. Ist der Zähler abgelaufen, wird ein Interrupt ausgelöst.

(2) Kernel Level:
Im Betriebssystemkern werden Zeiten insbesondere auf Basis eines zyklischen Timer Interrupts verarbeitet. Dabei zählt das Betriebssystem diese Interrupts (in Linux sind dies die sogenannten Jiffies) und bietet eine Reihe auf diesen Timerticks basierenden zeitgesteuerten Diensten an:

- Gerätetreiber können Tasks für eine wählbare Zeitdauer in den Zustand "wartend" versetzen.
- Module können zyklisch Funktionen (Threads) aufrufen lassen.
- Module können einmalig (zu einem Relativzeitpunkt) Funktionen aufrufen lassen.

Vorsicht ist geboten, wenn innerhalb eines (Betriebssystem-) Moduls Absolutzeiten benötigt werden. Der Abstand zwischen zwei Timerticks muss nicht zwangsläufig in jedem System identisch sein, sondern kann variieren.

Weiterhin kritisch ist der Umstand, dass Zähler eine endliche Breite haben und damit nach endlicher Zeit ein Zählerüberlauf stattfindet (in Linux beispielsweise nach 417 Tagen). Dieser Zeitpunkt kann für ein System kritisch sein, wenn nicht an allen Stellen (sowohl im eigentlichen Kernel als auch in den Treibern und Modulen) auf Zählerüberlauf geprüft wird.

(3) User Level:
An der Programmierschnittstelle lässt sich in UNIX-Betriebssystemen und deren Derivaten über die Funktion „gettimeofday" die Absolutzeit bestimmen. Um einen Prozess oder einen Thread für eine definierte Zeit in den Zustand „wartend" zu versetzen, existieren die Funktionen „sleep" und „select".

An der Dienstschnittstelle stellt das Betriebssystem für periodische Aufgaben „cron" zur Verfügung. Um eine Task zu einem bestimmten (Absolut-) Zeitpunkt zu starten, gibt es das Kommando „at". Die Programme „cron" und „at" dienen zur zeitverzögerten oder regelmäßig wiederkehrenden Ausführung von Befehlen. In Echtzeitsystemen stehen oftmals noch zusätzliche Hardware Timer zur Verfügung. Diese ermöglichen eine zeitgenauere Steuerung von Prozessen, als dies über das System bzw. über Software Timer möglich ist.

Während auf der Kernelebene Relativzeiten wegen der möglichen Zählerüberläufe kritisch sind, stellen auf der Applikations- bzw. Userebene Absolutzeiten eine Herausforderung dar.

3.3.5 Prozesse

Logische Struktur von Betriebssystemen

Wie wir bereits erfahren haben, wird das Betriebssystem zur logischen Strukturierung meist in mehrere Schichten eingeteilt. Die unterste Schicht, der sogenannte Betriebssystemkern (Kernel) beinhaltet alle hardwareabhängigen Teile des Betriebssystems, insbesondere auch die Verarbeitung von Unterbrechungen (engl. interrupts). Auf diese Weise ist es möglich, das Betriebssystem leicht an unterschiedliche Rechner mit unterschiedlichen Ressourcenausstattungen anzupassen. Die nächste Schicht enthält die grundlegenden *I/O-Dienste* für Plattenspeicher und Peripheriegeräte. Die darauffolgende Schicht behandelt *Kommunikations- und Netzwerkdienste, Dateien und Dateisysteme.* Weitere Schichten können je nach Anforderung folgen.

Nebenläufigkeit

Die meisten Applikationen im Bereich eingebetteter Systeme erfordern ein hohes Maß an nebenläufiger Verarbeitung sowie kurze Reaktionszeiten auf externe Ereignisse aufgrund harter Echtzeitanforderungen. Um die Entwicklungszeiten solcher Echtzeitsysteme so kurz wie möglich zu halten, werden hier Multitasking Betriebssysteme bzw. sogenannte Real Time Kernels eingesetzt. Ein Echtzeitbetriebssystem muss vom Entwickler in der Regel lediglich auf die entsprechende Zielhardware angepasst werden. Es ermöglicht ein flexibles Aufteilen von Systemressourcen (CPU, Speicher, etc.) auf einzelne Tasks bzw. Prozesse.

Prozesse, Tasks

Wie bereits festgestellt, verwalten Betriebssysteme Hardware-Ressourcen sowie Prozesse. Insbesondere das zeitgenaue Management von Prozessen erfordert bei Echtzeitbetriebssystemen eine besonders genaue Beachtung. Ein *Prozess* (auch *Task,* engl. für Aufgabe) ist ein ausführbares Programm plus die dazu gehörenden aktuellen Werte des Programm Counters (PC), der Register, Variablen und sonstigen Betriebsmittel (Speicher, I/O, Signale, usw.). Für die Implementierung eines Tasks hält das Betriebssystem eine eigene Tabelle bereit, den sogenannten *Task Control Block (TCB)* oder *Process Control Block (PCB).* Der PCB enthält die Prozessinformation sowie den CPU- und Systemkontext für den jeweiligen Task. Die Prozessinformation wiederum definiert den Prozesszustand (Beispiele: erzeugt, lauffähig, laufend, suspendiert,

terminiert) und die *Prozesspriorität* (Beispiele: hoch, mittel, niedrig).

Prozesszustände

Mögliche *Prozesszustände* sind im Wesentlichen (hier muss ergänzend hinzugefügt werden, dass Anzahl und Bennennung der Prozesszustände im Allgemeinen abhängig vom konkreten Betriebssystem sind):

- **Generiert:** Die Task wurde erzeugt.
- **Bereit** bzw. **lauffähig:** Eine Task ist bereit, wenn sie alle außer der CPU angeforderten Betriebsmittel zugeteilt bekommen hat.
- **Aktiv:** Eine Task ist aktiv, wenn sie die CPU gerade benutzt.
- **Blockiert** bzw. **suspendiert:** Eine Task ist blockiert bzw. suspendiert, wenn sie ihre Operation nicht ausführen kann, weil sie auf die Zuteilung mindestens eines noch angeforderten Betriebsmittels wartet.
- **Terminiert:** Eine Task ist beendet, wenn sie alle Anweisungen abgearbeitet und die ihr zugeteilten Betriebsmittel wieder freigegeben hat.

3.3.6 Multitasking und Scheduling

In Multitasking Betriebssystemen ist ein spezieller Systemprozess erforderlich, der aus den bereiten Tasks die nächste „aktive" Task auswählt. Diesen Prozess bezeichnet man als Scheduler. Sobald mehr als eine Task den Zustand „bereit" besitzt, muss der Scheduler durch Anwendung eines Schedulingalgorithmus festlegen, welche Task die CPU erhält.

Eigenschaften von Schedulern

Ein Scheduler sollte folgende *Eigenschaften* erfüllen:

- **Fairness:** Jeder Prozess erhält einen „fairen" CPU-Anteil.
- **Effizienz:** Die CPU sollte stets maximal ausgelastet sein.
- **Antwortzeit:** Interaktive Benutzer sollten so wenig lange wie möglich warten müssen.
- **Verweilzeit:** Stapelaufträge (Batchbetrieb) sollten eine so geringe Verweilzeit im Stapel haben wie möglich.
- **Durchsatz***:* Möglichst viele Aufträge sollten in einer bestimmten Zeitspanne bearbeitet werden.

- **Terminerfüllung:** Manche Ergebnisse müssen rechtzeitig zu festgelegten Zeitpunkten zur Verfügung gestellt werden.

Bei Multitasking Betriebssystemen werden zwei grundsätzlich unterschiedliche *Arten des Schedulings* unterschieden:

Kooperatives Multitasking

(1) Kooperatives Multitasking (engl. non preemptive multitasking): Die aktuell aktive Task gibt von sich aus freiwillig die CPU zu einem für sie geeigneten Zeitpunkt frei und eine andere Task kann diese bei Bedarf nun nutzen. In diesem Fall ist nur ein geringer Verwaltungsaufwand seitens des Schedulers nötig. Es besteht jedoch die Gefahr, dass ein „unkooperativer" oder fehlerhafter Prozess alle anderen Prozesse auf unbestimmte Zeit blockieren könnte.

Verdrängendes Multitasking

(2) Verdrängendes Multitasking (engl. preemptive multitasking): Das Gegenteil des kooperativen Multitasking ist das verdrängende Multitasking. Gibt es einen lauffähigen Task mit einer höheren Priorität als die des eben laufenden Tasks, so wird diesem die CPU entzogen obwohl dieser eventuell noch nicht vollständig abgearbeitet wurde. Der Scheduler kann einem solchen Prozess die CPU entziehen (z. B. ausgelöst durch den Ablauf einer Uhr, einem sogenannten Timer Interrupt). Dadurch kann die Bearbeitung dringlicherer Tasks mit höherer Priorität jederzeit begonnen werden. Ein fehlerhafter Prozess kann das System nicht blockieren. Hier handelt es sich um die in Echtzeitbetriebssystemen bevorzugte Variante.

Der Anstoß für den Prozesswechsel durch eine Verdrängung erfolgt in Abhängigkeit von der Scheduling Strategie. Man kann im Wesentlichen die *folgenden grundlegenden Strategien* unterscheiden:

Grundlegende Scheduling Strategien

- **Zeitgesteuerte Strategien:** Jeder Prozess erhält die CPU für eine bestimmte Zeitspanne, die sogenannte Zeitscheibe (engl. time slice). Danach wird die CPU dem nächsten Prozess zugeteilt. Man spricht hier von der sogenannten Zeitscheibensteuerung bzw. Round Robin Verfahren.
- **Ereignisgesteuerte Strategien:** Ein Prozesswechsel findet dann statt, wenn ein Ereignis (z. B. ein Hardware Interrupt) einen anderen Prozess benötigt. Hier werden allgemein den einzelnen Prozessen Prioritäten zugeordnet, die sich dynamisch ändern können. Ein bestimmtes Ereignis verleiht dem gewünschten Prozess eine höhere Priorität.

Sowohl beim kooperativen Multitasking als auch bei den ereignisgesteuerten Scheduling Strategien wird die Zuteilung mit Hilfe von Taskprioritäten gesteuert. Hier spricht man beim kooperativen System von *relativem Vorrang* (der erst zum nächstmöglichen Zeitpunkt nach Freigabe der CPU durch den aktiven Prozess wirksam wird) und beim ereignisgesteuerten System *vom absoluten Vorrang* (der sofort zu einem Prozesswechsel führt). Die Zeitscheibensteuerung kann als Sonderfall der Ereignissteuerung betrachtet werden, das entsprechende Ereignis wird in diesem Fall durch den Ablauf der zugeteilten Zeitscheibe ausgelöst.

Einige bewährte *Scheduling Strategien,* die in der Praxis verwendet werden, sind:

Bewährte Scheduling Strategien aus der Praxis

- **First Come, First Served (FCFS):** Die Verteilung der Prioritäten erfolgt strikt nach Ankunftszeit, d. h. dem Zeitpunkt, wenn die Task erzeugt wird und so erstmals den Status „generiert" erhält. In dem Fall, dass zwei oder mehr Prozesse genau gleichzeitig diesen Status erhalten, wird eine zufällige Auswahl getroffen. Diese Strategie zeichnet sich durch eine gute Systemauslastung aus, hat aber ein schlechtes Antwortzeitverhalten, so dass langlaufende Prozesse (also solche, die über eine längere Zeit hinweg den Status „aktiv" einnehmen) Kurzläufer behindern. FCFS ist ein vergleichsweise einfach zu implementierendes Verfahren.
- **Zeitscheiben Verfahren (Round Robin):** Jedem Prozess wird eine feste Zeitspanne (auch: Zeitscheibe) zugeordnet. Die Zeitscheibe (engl. time slice) wird dabei so kurz gewählt, dass keine bemerkbaren Verzögerungen auftreten und es für den Benutzer den Anschein hat, ihm bzw. seinem Task „gehöre" die CPU alleine. Nach Ablauf dieser Zeitspanne wird er verdrängt und der nächste Prozess erhält die CPU. Insgesamt ergibt sich auf lange Sicht eine zyklische Zuteilung (daher auch der Namensteil „round"). Alle Prozesse haben über ihre komplette Lebenszeit hinweg eine gleichbleibende Priorität. Die Zeitscheibe kann konstant sein oder abhängig von der Prozessorbelastung variieren. Bei kleinen Zeitscheiben lassen sich vergleichsweise kurze Antwortzeiten in Verbindung mit höheren Verlusten durch die häufigen Prozesswechsel erreichen. Beispiel: Windows bis Version 3.0.
- **Prioritätssteuerung:** Jedem bereiten Prozess wird eine Priorität zugeordnet. Die Vergabe der CPU erfolgt in absteigender Priorität. Ein Prozess niedrigerer Priorität kann die CPU erst dann er-

halten, wenn alle Prozesse mit einer höheren Priorität abgearbeitet wurden. Ein Prozess mit höherer Priorität, der gerade den Zustand „bereit“ erhält, verdrängt daher stets einen aktiven Prozess niedrigerer Priorität. Alle Prozesse gleicher Priorität werden im Allgemeinen in jeweils einer eigenen Warteschlange zwischengespeichert. Zur Prioritätssteuerung gibt es verschiedene, teilweise gemischte Detailverfahren, so z. B.:

Spezielle Prioritätssteuerungen

- *Reine Prioritätssteuerung:* Alle Prozesse gleicher Priorität werden nach deren Eingangsreihenfolge abgearbeitet. Hier handelt es sich um einen oft in Echtzeitbetriebssystemen verwendeten Ansatz.
- *Prioritätssteuerung mit unterlegtem Zeitscheiben Verfahren:* Sämtliche Prozesse gleicher Priorität werden nach dem Zeitscheiben Verfahren abgearbeitet. Beispiel: UNIX.
- *Dynamische Prioritätsvergabe:* Die Priorität der auf die Zuteilung der CPU wartenden Prozesse wird mit zunehmender Wartezeit allmählich erhöht.
- *Mehrstufiges Herabsetzen (Multilevel Feedback):* Eine maximale Zuteilungszeit der CPU wird für alle Tasks einer Prioritätsstufe festgelegt. Dies geschieht für alle Prioritätsstufen getrennt. Hat ein Prozess die maximale Rechenzeit seiner Priorität verbraucht, wird ihm sodann die nächstniedrigere Priorität zugewiesen, bis er ggf. die niedrigste Prioritätsstufe erreicht.

Darüber hinaus gibt es noch zahlreiche weitere Scheduling Strategien, die in diesem Rahmen nicht aufgeführt werden können (Tanenbaum, 2001). In Dialogsystemen (also beispielsweise in Betriebssystemen in PCs) wird normalerweise das Round Robin Verfahren verwendet, um allen Benutzern akzeptable Antwortzeiten zu bieten. Bei Stapelbetrieb und gemischten Systemen kommen oft Kombinationen der vorgenannten Strategien vor, z. B. getrenntes Scheduling für Dialog- und Batch Betrieb.

3.3.7 Scheduling in Echtzeitbetriebssystemen

Echtzeitbetriebssysteme sind durchwegs Multitasking-fähige Betriebssysteme. Alle Tasks können unabhängig voneinander gestartet und ausgeführt werden. Sie werden durch Unterbrechungen (engl. interrupts), Zeitabläufe (engl. timeouts) und Eingabeereignisse

(engl. input events) gesteuert. Diese Tasks müssen miteinander kommunizieren und sich koordinieren, um gemeinsame Aufgaben lösen zu können. Der Scheduler ist die zentrale Instanz eines Echtzeitbetriebssystems. Er entscheidet, wann eine Task den Prozessor zugesprochen bekommt und wann sie ihn wieder abgeben muss. Hierzu gibt es verschiedene Schedulingstrategien, die unterschiedlich gut für Echtzeitbetriebssysteme geeignet sind.

In marktüblichen Mehrbenutzerbetriebssystemen sind die im vorangegangenen Abschnitt skizzierten Scheduling Strategien FCFS oder Round Robin ausreichend. Dies ist für Echtzeitbetriebssysteme jedoch nicht mehr der Fall. Hier sind besondere Strategien erforderlich, welche den Echtzeitanforderungen besonders Rechnung tragen.

EDF-Verfahren

Ein erster Ansatz führt uns zum *Earliest Deadline First Scheduling-Algorithmus* (kurz EDF). Beim EDF wird derjenige Task mit der nächstliegenden Deadline ausgeführt. EDF ist optimal in dem Sinne, dass er für jedes schedulebare Prozesssystem einen zulässigen Schedule konstruiert (Fränzle, 2002). Dies kann durch einen formalen Beweis belegt werden.

Trotz dieser Optimalitätsergebnisse wird EDF in der Praxis selten eingesetzt, da es deutliche *Nachteile* gegenüber anderen Verfahren hat (Fränzle, 2002): Falls sporadische (also nicht vor der Laufzeit bekannte) Tasks im System enthalten sind, können mit EDF festgelegte Schedules nicht vorab berechnet werden. Stattdessen wird eine dynamische Sortierung der Tasks während der Laufzeit erforderlich, die vergleichsweise aufwändig ist und damit die Annahme unrealistisch macht, dass Kontextwechselzeiten und Scheduling Overhead vernachlässigbar sind. EDF neigt darüber hinaus zu überflüssigen Kontextwechseln, da es einen laufenden Prozess auch dann sofort zu Gunsten eines Prozesses mit kürzerer Deadline unterbricht, wenn die weitere Bearbeitung dieses Prozesses problemlos möglich gewesen wäre. Diese Vorgehensweise erzeugt unnötigen Overhead und wirkt sich somit negativ auf die Laufzeit aus.

Prioritätensteuerung

Um das Problem der dynamischen Prozessumordnung zu verringern, verwendet man in der Regel durch *Prioritäten* gesteuerte Scheduler mit fest vergebenen Prioritäten (siehe auch Abschnitt 3.3.6). In einem solchem System ist jedem Prozess eine Priorität fest zugeordnet und zu jedem Zeitpunkt wird der gerade lauffähige Prozess mit der höchsten Priorität ausgeführt. Manche Tasks und Ereignisse sind wichtiger als andere (haben also eine höhere Priorität) und sind daher vorrangig zu behandeln. Um dieser Anforderung Rechnung zu

tragen, arbeiten Echtzeitbetriebssysteme deshalb mit einem anderen Verfahren, der sogenannten *Prioritätensteuerung*.

Der Programmierer gibt jeder Task eine feste d. h. statische Priorität, und der Scheduler wählt jeweils die lauffähige Task mit der höchsten Priorität für die CPU-Zuteilung aus. Auf diese Weise werden hoch priorisierte Aufgaben vorrangig erledigt und die weniger wichtigen unterbrochen. Nur wenn mehrere Tasks mit gleicher Priorität vorliegen, greift der Scheduler wieder auf ein anderes Verfahren, beispielsweise Round Robin zurück.

Um eine statische Festlegung (also vor der Laufzeit) der Priorität treffen zu können, müssen alle Prozessparameter vorab bekannt sein. Der Vorteil dieses Konzepts liegt in der Ausführungsgeschwindigkeit: Zur Laufzeit ist kein Planungsaufwand erforderlich. Das dynamische Scheduling ist dagegen zwar flexibler, weil alle Planungsentscheidungen erst zur Laufzeit getroffen werden müssen, benötigt aber mehr Planungszeit und ist daher für Echtzeitbetriebssysteme nicht geeignet.

RMS

Die resultierende Problematik besteht darin, eine sachgerechte Prioritätenzuordnung zu finden (Fränzle, 2002). Ein bewährter Algorithmus hierfür ist das *ratenmonotone Scheduling (RMS):*

Die *Rate* eines periodischen Prozesses ist der Kehrwert seiner Periode. Bei sporadischen Prozessen bezieht man sich ersatzweise auf die Maximalrate, die sich als reziproker Wert des Minimalabstandes ergibt. Ein ratenmonotoner Schedule entsteht, wenn in einem präemptiven Multitaskingsystem stets der lauffähige Prozess mit der höchsten Priorität ausgeführt wird, wobei die Priorisierung zwischen den Prozessen der Ratenordnung entspricht. Falls der ratenmonotone Schedule eines Prozesssystems für jede bezüglich der Prozessparameter mögliche Aktivierungssequenz zulässig ist, bezeichnen wir das Prozesssystem als ratenmonoton schedulebar, kurz RMS-schedulebar. Der Scheduler kann immer effektiv prüfen, ob ein Prozesssystem RMS-schedulebar ist. Ein Nachweis ist durch formalen Beweis möglich.

3.3.8 Speicherverwaltung

Im Multitaskingbetrieb brauchen alle verwalteten Tasks einen zugeteilten Speicher, um lauffähig zu sein. Zusätzlich zur Prozessverwaltung (Scheduling) ist also eine Speicherverwaltung erforderlich. Wie sich der Leser vielleicht noch erinnern mag, kommt hier in der Regel eine Speicherhierarchie (Register, Cache, Hauptspeicher und Festplatte) zum Einsatz. Auch im Falle des

Speichermanagements können Standardverfahren anderer Betriebssysteme nicht ohne weiteres übernommen werden.

Paging, Swapping

Die meisten Betriebssysteme ermöglichen es Applikationen, mehr (virtuellen) Speicher zu verwenden als tatsächlich physikalisch vorhanden ist. Die hierzu verwendeten Techniken heißen Paging und Swapping. Beim *Paging* werden im Bedarfsfall Teile (Blöcke) des Hauptspeichers, Seiten (engl. pages) genannt, auf eine Festplatte transferiert und bei Zugriff einer Applikation wieder zurück geschrieben. Das Ein- und Auslagern ganzer Prozesse bezeichnet man als *Swapping*. Dieses Konzept ist heute allerdings nur noch selten in Benutzung. Aktuelle Betriebssysteme verwenden das Paging. Beiden ist allerdings gemein, dass ihr Einsatz zu einem nichtdeterministischen Systemverhalten führen kann.

Sollen Echtzeitanforderungen erfüllt werden, so kann es ggf. zu lange dauern, wenn erst zur Laufzeit Programmcodes oder Daten von der Festplatte in den Hauptspeicher nachgeladen werden müssen. Dynamisches Nachladen von Speicherbereichen aus einem virtuellen Speicher, wie es z. B. bei Unix oder Linux üblich ist, kann daher im Allgemeinen zur Realisierung von Echtzeitaufgaben nicht verwendet werden. Echtzeitbetriebssysteme, die tatsächlich Swapping oder Paging anbieten, sehen dies nur für zeitunkritische Tasks vor.

In einem Echtzeitbetriebssystem müssen alle (echt-) zeitkritischen Tasks im Hauptspeicher gehalten werden. Somit kommt der Speicherverwaltung eine sehr wichtige Rolle zu. Dabei muss bedacht werden, dass jederzeit ein Task mit einer Speicheranforderung oder Speicherfreigabe an die Speicherverwaltung herantreten kann, dass rekursive Strukturen mit lokalen Datenbereichen möglich und auch üblich sind, dass gleiche Tasks auch mehrmals parallel laufen können (ggf. mit Code-Sharing), dass sogar während der Allokation von Speicher eine höher priorisierte Task „dazwischenfunken“ kann und vieles mehr. Quasi beiläufig muss das Speichermanagement auch noch Lücken im Speicher optimal zusammenfassen und nicht mehr benötigte Daten entfernen (Carbage Collection, vgl. Java). Der Speichermanager bestimmt nicht zuletzt die Reaktionszeit auf ein Ereignis. Bei Echtzeitbetriebssystemen wird die Reaktionszeit umso länger, je mehr Tasks sich gleichzeitig im Hauptspeicher befinden, weil der Speichermanager bei der Allokation mehr Zeit benötigt, um freien Speicher zu finden (auch hier gibt es natürlich Allokationsstrategien, um die Suchzeit so gering wie möglich zu halten).

Caching

Moderne Prozessoren haben Caches eingebaut, durch die die Performanz des Systems erheblich gesteigert wird. Caches induzieren allerdings ein nicht.deterministisches Verhalten. Es lässt

sich nicht vorhersagen, ob der nächste Programmbefehl oder das nächste benötigte Datum sich im Cache befindet oder nicht. Der Entwickler aber muss wissen, dass andererseits das Abschalten von Caches in der Regel zu teilweise erheblichen Performanzverlusten führen kann.

3.4 VxWorks als Beispiel eines Echtzeitbetriebssystems

Im folgenden wollen wir nun das Echtzeitbetriebssystem VxWorks (Windriver, 1993) der Firma Wind River Systems Inc. etwas genauer betrachten. VxWorks wurde unter anderem bei der US-amerikanischen Pathfinder-Mission 1996 zum Mars eingesetzt und wird derzeit in der Formel 1 als Betriebssystem in den Rennfahrzeugen verwendet, um mit Hilfe von Anwendungen die Kommunikation und Interaktion zwischen Fahrer und Rennstall zu realisieren (Quelle: www.about-it.de).

VxWorks ist ein speziell für Steuerungs- und Datenerfassungszwecke entwickeltes Echtzeitbetriebssystem. Anders als bei Windows NT/2000/XP und UNIX-Derivaten, gibt es bei VxWorks keinen Benutzerbereich, d. h. einen Bereich, der deutlich vom System abgegrenzt ist und nur begrenzte direkte Zugriffsrechte auf Systemressourcen erlaubt. Erweiterungen, um mit Hardware zu kommunizieren, werden direkt in den Kernel geladen. Direkte Hardware-Zugriffe stellen somit in VxWorks kein Problem dar (Breuer et al., 2002).

VxWorks ist aus verschiedenen Komponenten aufgebaut; die *drei Hauptkomponenten* sind ein skalierbares *Laufzeitsystem* für ein Zielsystem (siehe Abbildung 3.1), ein *Netzwerkmodul für verteilte Systeme* (mit einer TCP/IP- und Ethernet-Unterstützung) sowie der *Entwicklungsumgebung* Tornado. VxWorks besteht aus einem Kernel, der in seinem Aufbau und seiner Funktionalität einem UNIX Kernel sehr ähnlich ist sowie zusätzlichen Funktionen bzw. Bibliotheken. Dienste befinden sich auf der Taskebene statt im Kernel. VxWorks kann seit 1995 parallel zu Windows auf PCs installiert werden. Dies bietet die komfortable Möglichkeit, Windows für die Bedienoberfläche und Auswertung zu verwenden, während VxWorks die Steuerung und Regelung von Systemen übernimmt.

VxWorks besitzt einen Hochleistungs-Mikrobetriebssystemkern. Dieser Microkernel unterstützt eine Vielzahl von Echtzeitfunktionen einschließlich schnellem Multitasking, Interrupt Support, Pre-

emptive und Round Robin Scheduling. Der besondere Aufbau des Microkernels macht es möglich, die Auslastung des Systems so gering wie möglich zu halten und gleichzeitig auf äußere Ereignisse schnell und deterministisch zu reagieren. VxWorks ist sehr flexibel. Es kann sowohl in einer minimalen Konfiguration eines Kernels mit nur wenigen Modulen für eingebettete Systeme als auch in einer voll ausgebauten Variante, welche viele Benutzer bedient, betrieben werden. Die hierfür tatsächlich benötigten Ressourcen sind zu installieren. Entwickler können VxWorks sehr einfach an die Bedürfnisse ihrer eigenen Applikationen anpassen. Weiterhin besitzt VxWorks einen effektiven Intertask Mechanismus der es unabhängigen Tasks erlaubt, ihre Aktionen innerhalb des Echtzeitsystems selber zu koordinieren.

Auf Grund dieser Eigenschaften und weiterer wie etwa Multiprozessorfähigkeit, Shared Memory, Unterstützung des VME-Bus, Message Queues etc. ist VxWorks besonders gut für den Einsatz im Bereich eingebetteter Systeme geeignet. Ein besonderer Vorteil dieses Betriebssystems ist die Möglichkeit, kompilierte Objektdateien beim Laden dynamisch zu linken. Dies bedeutet, dass zur Laufzeit eine Objektdatei geladen werden kann und alle darin enthaltenen globalen Variablen und Symbole stehen danach betriebssystemweit zur Verfügung. Ein weiterer entscheidender Vorteil von VxWorks ist dessen Skalierbarkeit bzw. Konfigurierbarkeit. Dies macht das Betriebssystem für den universellen Einsatz geeignet. So kann VxWorks sowohl in Applikationen zum Einsatz kommen, die zur Laufzeit nur einen kleinen effizienten Betriebssystem Kernel benötigen, als auch bei großen verteilten Systemen, die UNIX-kompatible Schnittstellen benötigen.

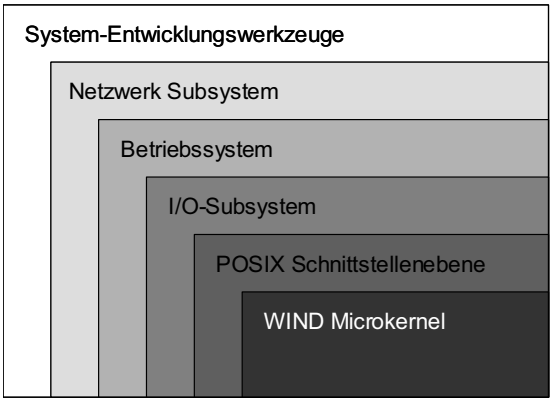

Abb. 3.1: Das skalierbare Laufzeit-System von VxWorks (aus (Berthold, 1996))

3.4.1
Das Laufzeitsystem

Die Basis des VxWorks Betriebssystems ist der sogenannte „Wind"-Microkernel (siehe Abbildung 3.1). Durch diesen Kernel werden eine Multitasking-Umgebung sowie Mechanismen zur Interprozess-Kommunikation und Synchronisation zur Verfügung gestellt. Alle anderen Betriebssystemaufgaben werden auf die Ebene der Tasks verlagert und auf die einfache Kernel-Basis abgebildet. Eine wichtige Aufgabe des Kernels besteht in der Bereitstellung einer Umgebung zur Verwaltung nebenläufiger Tasks. Der Echtzeit-Kernel ist für die deterministische Zuteilung von Rechenzeit auf der CPU für die verschiedenen Tasks zuständig. Dies wird im Gegensatz zu Unix-Systemen mit einem prioritätsbasierten, unterbrechenden Multitasking realisiert. Insgesamt stehen 256 unterschiedliche Prioritätsstufen zur Verfügung. Jeder Stufe können prinzipiell beliebig viele Tasks zugeteilt werden. Zur Synchronisierung und Kommunikation zwischen den verschiedenen Tasks wird das Konzept der (binären sowie zählenden) Semaphore und Message Queues verwendet. Um Plattform-Unabhängigkeit und Portabilität zu gewährleisten, ist der Kernel über eine Schnittstelle nach POSIX 1003.1b (vgl. www.posix.com) zugänglich gemacht.

3.4.2
Exkurs: Der POSIX Standard

POSIX, das „Portable Operating System Interface", ist eine Spezifikation, die vom IEEE erarbeitet und von ANSI und ISO standardisiert wurde. Aufgabe des POSIX Standards ist es, die Portabilität von Applikationen auf Quellcodeebene zu gewährleisten. Im Rahmen von POSIX werden eindeutige Schnittstellen in Form von Funktionen definiert, die ein Betriebssystem bereitstellen muss, um diesem Standard zu genügen. Ein POSIX konformer Quellcode ist daher auf allen POSIX Betriebssystemen kompilierbar. Beim *POSIX Substandard IEEE 1003.1b* handelt es sich um echtzeit-spezifische Erweiterungen sowie I/O-Erweiterungen. POSIX 1003.1b erlaubt die Verwendung eines Betriebssystems in *weichen* Echtzeit-Situationen. Ferner können als Scheduling Strategien Round Robin sowie ein statisches Prioritätenverfahren verwendet werden.

3.4.3 Das I/O-Subsystem von VxWorks

Für eingebettete Systeme ist der Zugriff auf Hardwarekomponenten wesentlich. In VxWorks kann Speicher direkt adressiert werden (engl. Direct Memory Access, kurz: DMA). Somit kann auch unmittelbar auf Hardware zugegriffen werden, was die Verwaltung selbst sehr zeitkritischer Hardwarekomponenten ermöglicht. Darüber hinaus stellt VxWorks ein komplettes Unix-kompatibles I/O-System zur Verfügung. Treiber können so nach bekannten Mechanismen in das System – sogar dynamisch während der Laufzeit – eingebunden werden. Ferner können weitere Unix-Treiber leicht nach VxWorks portiert werden.

3.4.4 Unterstützung verteilter Systeme in VxWorks

Eine stets anwachsende Zahl eingebetteter Systeme ist Teil komplexer verteilter Systeme, bei denen die Kommunikation untereinander eine entscheidende, weil zeitkritische Rolle spielt. Klassische gängige Kommunikationsprotokolle erfüllen jedoch keine Echtzeitanforderungen. VxWorks stellt echtzeitfähige, aber auf Standard-Unix basierende Kommunikationsprotokolle zur Verfügung.

3.4.5 VxWorks Entwicklungswerkzeuge

Um echtzeitfähige Anwendungen entwickeln zu können, müssen dem Entwickler passende Entwicklungswerkzeuge und -konzepte zur Verfügung gestellt werden. Das Cross-Entwicklungspaket Tornado für VxWorks ist auf die speziellen Anforderungen der Softwareentwicklung im Bereich der Echtzeitsysteme maßgeschneidert. Die Programmentwicklung (Spezifikation, Codierung, Kompilierung und Debugging) erfolgt auf einem UNIX-Rechner. Abbildung 3.2 verschafft einen Überblick.
Um Portabilität eines Programms auf verschiedensten eingebetteten Systemen zu gewährleisten, sollte stets der gleiche Compiler verwendet werden. Im Falle von VxWorks kommt hier der GNU-C (vgl. www.gnu.org) bzw. C^{++}-Compiler zum Einsatz. Der Cross-Debugger GNU-GDB ermöglicht dann ein Source Level Debugging. VxWorks bietet ferner über eine Shell (eine Shell ist eine flexible,

Kommandozeilen-basierte Benutzerschnittstelle des Betriebssystems, welche vorzugsweise zum Start von Prozessen verwendet wird) auf dem Entwicklungsrechner einen interaktiven Zugriff auf den Zielrechner. Auf diese Weise können alle C-Funktionen, die der Anwender mit seinen Anwendungsmodulen auf den Zielrechner überträgt, aufgerufen und getestet werden. Darüber hinaus besteht die Möglichkeit, mit Hilfe der Shell Betriebssystemroutinen aufzurufen und auszuführen. VxWorks bietet weiterhin die Möglichkeit, interaktiv Softwaremodule auf das Zielsystem schrittweise nachzuladen. Ein neues Modul muss lediglich neu kompiliert, nicht aber neu gelinkt werden. Offene Referenzen werden dabei über Symboltabellen auf dem Ziel- und dem Hostsystem aufgelöst. Mit einem System-Browser erfolgt eine graphische Darstellung des resultierenden Echtzeitsystems. Dabei lassen sich Speicherbereiche oder Informationen über Systemobjekte (Tasks, Semaphore, Warteschlangen) abfragen und darstellen.

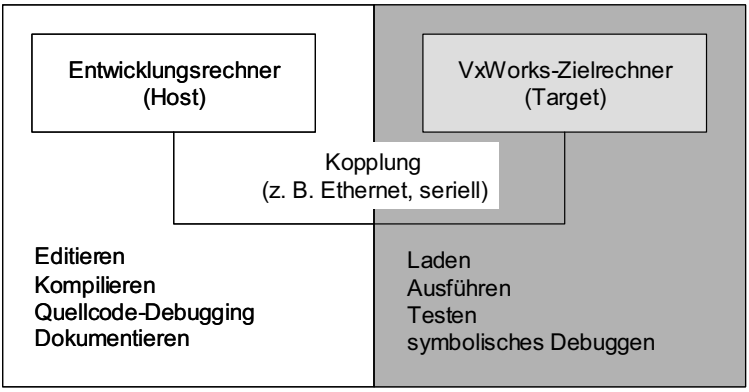

Abb. 3.2: Das Konzept der Cross-Entwicklung in VxWorks (aus (Berthold, 1996))

Programmierer können bei der Entwicklung ihrer Programme mehrere Methoden des Betriebssystems nutzen. Unter anderem gehören Shared Memory, Message Queues, Semaphoren, Events und Pipes (für die Intertask Kommunikation innerhalb einer CPU), Sockets und Remote Methoden dazu. Außerdem steht dem Benutzer eine Vielzahl von Industriestandards zur Verfügung wie z. B. POSIX, TCP/IP, UDP, ARP, RIP v1/v2, SLIP, CSLIP, BOOTP, DNS, DHCP, TFTP, NFS, SUN RPC, FTP, rlogin, rsh, telnet, SNTP usw. Als Programmiersprache steht neben C und C++ auch Java zur Verfügung.

Ständig aktualisierte Informationen über VxWorks finden sich im Internet auf den Seiten der Firma WindRiver Systems Inc. (www.wrs.com).

3.5 Weitere Beispiele eingebetteter Betriebssysteme

Derzeit gibt es über 100 verschiedene eingebettete Betriebssysteme; aber nicht jedes von ihnen ist für einen bestimmten Mikroprozessor geeignet, da hierzu Betriebssystem-spezifisch einige Grundvoraussetzungen erfüllt sein müssen. So benötigt z. B. das Betriebssystem Windows CE 5.0 eine 32 Bit CPU (ARM, SHx, MIPS oder x86), 250 KByte Flash und mindestens 6 bis 8 MByte RAM. Hat man sich mal für einen bestimmten Mikroprozessor entschieden, kann man meist immer noch zwischen mehr als einem Dutzend Betriebssystemen auswählen.

Mobile Systeme, mobile Betriebssysteme

Die nachfolgende Auswahl von eingebetteten Betriebssystemen erhebt folglich keinen Anspruch auf Vollständigkeit. An dieser Stelle sei darauf hingewiesen, dass es sich bei vielen dieser eingebetteten Betriebssysteme um keine Echtzeitbetriebssysteme im engeren Sinne, sondern um Betriebssysteme für mobile Rechnersysteme handelt. Unter dem Terminus *„mobile (Rechner-)Systeme“* wollen wir hier nur solche Systeme zusammenfassen, die von ihrem Benutzer mitgeführt und unabhängig von anderen Systemen verwendet werden (z. B. Mobiltelefone, PDAs, Handhelds, Smartphones, Wearables usw.). Sie alle teilen die Eigenschaft, nur weichen, aber keinen harten Echtzeitanforderungen genügen zu müssen. Folglich müssen die meisten mobilen Betriebssysteme ebenfalls keine harten Echtzeitanforderungen erfüllen. Unter einem *mobilen Betriebssystem* wird im Folgenden ein Programm für die Verwaltung der Ressourcen eines mobilen Rechnersystems verstanden.

Nichtsdestoweniger müssen auch mobile Betriebssysteme teilweise restriktiven Anforderungen an ihre Leistungsfähigkeit genügen. So unterliegen mobile Systeme oft Beschränkungen wie leistungsschwachen Prozessoren, geringen Speicherressourcen, eingeschränkten Displaygrößen und Eingabemöglichkeiten sowie einer begrenzten Stromversorgung. Durch diese Restriktionen ist der Einsatz von UniversalBetriebssystemen selten möglich. Darüber hinaus werden an mobile Betriebssysteme Anforderungen gestellt, die sich an dem Mobilitätsgrad und dem gewählten Endgerätetyp orientieren (Schiefer, 2004), (Devooght, 2003):

- Ausfallsicherheit
- Schnelle Betriebsbereitschaft (kurze Ladezeiten)
- Ergonomie (Benutzerfreundlichkeit, intuitive Bedienbarkeit)
- Einhaltung weicher Echtzeitanforderungen

- Erweiterbarkeit und Anpassbarkeit für neue Funktionen und Kokmmunikationskanäle
- Hohe Inormationssicherheit
- Geringer Energieverbrauch durch eine ausgefeilte Energieversorgung (engl. power management), z. B. das Abschalten nicht verwendeter Systemkomponenten.

In diesem Abschnitt sollen einige weit verbreitete Betriebssysteme für mobile Systeme vorgestellt werden. Es wurden Symbian OS, Palm OS und Windows CE ausgewählt. Zur Gegenüberstellung mobiler Betriebssysteme und Echtzeitbetriebssysteme schließt dieser Abschnitt mit einer kurzen Betrachtung der beiden Echtzeitbetriebssysteme QNX und Embedded Linux.

3.5.1 Symbian OS

PDA

Symbian OS ist ein Betriebssystem für PDAs (Personal Digital Assistants) und Smartphones. Es stammt von Psions 32 Bit EPOC Plattform ab. Zahlreiche Kommunikationsprotokolle werden von ihr unterstützt. Schwerpunkte setzt Symbian dabei vor allem in der drahtlosen Kommunikation (WAP, Bluetooth usw.), im Netzwerkbereich und in der Telekommunikation. Symbian wurde 1998 gegründet und besteht aus folgendem Konsortium: Arima, benQ, Fujitsu, Lenovo, LG Electronics, Mitsubishi Electric, Nokia, Motorola, Panasonic, Sanyo, Sendo, Sony Ericsson, Psion und Siemens.

Robustheit

Das Symbian Betriebssystem wurde so konzipiert, dass Abstürze die einen Reboot hervorrufen, so gut wie unmöglich sind. Um dies zu erreichen, läuft jeder Prozess in einem eigenen, geschützten Adressbereich; so ist es einer Anwendung nicht möglich, den Adressbereich einer anderen Anwendung zu überschreiben. Auch der Kernel läuft analog in einem geschützten Adressbereich, so dass es für keine Anwendung möglich ist, den Stack oder den Heap des Kernels zu überschreiben und dadurch einen systemweiten Absturz zu verursachen.

Speicher

Symbian unterscheidet zwischen Arbeitsspeicher (RAM) und Festspeicher (Flash-Speicher, Erweiterungskarten). Um Arbeitsspeicher zu sparen, wird die sogenannte Execute-in-Place Methode verwendet: Anwendungen lassen sich direkt aus dem Flash-Speicher ausführen. Im Gegensatz zu vielen anderen vergleichbaren Betriebssystemen unterstützt Symbian präemptives Multitasking und

Multithreading, wodurch allerdings wiederum der Arbeitsspeicherbedarf erhöht wird.

Entwicklung

Anwendungen können in folgenden Sprachen implementiert werden: C++, Java, OPL und .NET. Hilfreiche Informationen für Entwickler u. a. finden sich unter www.symbian.com.

3.5.2 Palm OS

PDA

Das Palm OS gehört zu den am weitesten verbreiteten Vertretern von eingebetteten Betriebssystemen, da es v. a. in zahlreichen PDAs Einsatz findet. So wurden bisher rund 21 Millionen Geräte, welche auf Basis dieses Betriebssystems betrieben werden, ausgeliefert. Ferner stehen rund 13.000 Applikationen für das Palm OS zur Verfügung. Obwohl es vorzugsweise in PDAs verwendet wird, gibt es durchaus auch andere Gerätetypen, vor allem medizinische Instrumente und Smartphones, die es als Betriebssystem einsetzen.

Kernel

Das Palm OS ist wie viele andere Betriebssysteme auch um einen Kernel herum aufgebaut. Dieser Kernel stellt die Grund-funktionalität zur Verfügung. Fast alle höheren Funktionen werden dann durch einen sogenannten „Manager“ zur Verfügung gestellt. Will eine Applikation beispielsweise einen Alarm auslösen, so muss sie den entsprechenden Befehl an den Manager weiterleiten – erst dieser löst dann den Alarm aus. Dem Applikationsentwickler wird durch die Überwachungsfunktion des Managers einerseits eine hohe Abstraktionsebene geboten, andererseits bietet sie Schutz vor fehlerbehaftetem Code.

Multitasking

Verwirrung gibt es in den einschlägigen Foren oftmals hinsichtlich der Multitasking-Eigenschaften des Palm OS, darum sei hier folgendes klargestellt: Das Palm OS ist ein 32 Bit, ereignisgesteuertes Betriebssystem mit eingeschränkten Multitasking-Fähigkeiten. Auf Betriebssystemebene können nebenläufig mehrere Tasks ausgeführt werden, auf Anwendungsebene jedoch nicht.

Speicher-management

Das Palm OS benutzt den RAM als Arbeitsspeicher sowie zum permanenten Speichern von Daten und Programmen. Dabei wird der RAM in zwei Segmente unterteilt, in die sogenannte Storage Area sowie in die Dynamic Area. Die Größe der einzelnen Areas hängt vom Betriebssystem und natürlich vom verfügbaren Speicher im Gerät ab, kann also nicht eingestellt werden. Das Palm OS läßt eine Speichererweiterung durch Speicherkarten zu. Um Arbeitsspeicher zu sparen, wird wie beim Symbian OS die Execute-in-Place Methode verwendet.

3.5.3 Windows CE

Im Gegensatz zu den bis hierher vorgestellten Betriebssystemen hat man bei der Entwicklung von Microsoft Windows CE versucht, möglichst nahe an den bereits bestehenden Universalbetriebssystemem der Windows-Familie zu bleiben. Das Design ist daher zwangsläufig nicht so konsequent auf den Bereich eingebetteter Systeme zugeschnitten, wie dies bei den anderen Betriebssystemen der Fall war. So ist Windows CE beispielsweise ressourcenintensiver als VxWorks oder QNX. Andererseits findet der Benutzer eine vertraute Oberfläche vor. Windows CE besitzt eine offene Architektur und unterstützt daher eine Vielzahl von Hardwarekomponenten. Microsoft sieht das Hauptanwendungsgebiet von Windows CE im Informations-, Kommunikations- und Unterhaltungsbereich (Internet-TV, Set-Top-Boxen) (Quelle: www.microsoft.com).

Speicher-management

Speichermanagement:
Das 32 Bit Betriebssystem Windows CE verwaltet den Speicher ähnlich wie das Palm OS. Der RAM wird in zwei Teile aufgeteilt, ein Teil als Arbeitsspeicher, der andere als permanenter Speicher. Das System selber ist im ROM abgespeichert. Im Gegensatz zu den vorher vorgestellten Betriebssystemen wird bei Windows CE nur das Betriebssystem „in place“ ausgeführt. Applikationen werden in den Arbeitsspeicher kopiert und von dort ausgeführt. Ein weiterer sehr wichtiger Unterschied etwa zu Palm OS oder Symbian OS ist folgender: Daten, die im permanenten Speicher von Windows CE gehalten werden, sind komprimiert. Windows CE unterstützt eine große Anzahl von Speichererweiterungen, so dass auch größere Anwendungen bzw. Datenmengen auf einem Windows CE System abgelegt werden können.

Multitasking

Multitasking:
Windows CE arbeitet wie Windows NT, 2000 und XP mit präemptiven Multitasking. Um das System nicht zu überlasten, wird die Anzahl von gleichzeitig laufenden Applikationen auf maximal 32 limitiert. Wird diese Grenze erreicht, kann Windows CE eine Applikation schließen.

Programmieren

Programmieren unter Windows CE:
Das Studium der Entwicklungsumgebungen für Windows CE zeigt, dass Microsoft den Bereich der Programmiersprachen C/C++ und Basic mit seinen Embedded Visual Tools dominiert. Dies hat dem Anschein nach vornehmlich folgende Gründe. Zum einen sind die

Embedded Visual Tools ohne Aufpreis erhältlich, zum anderen sind sie sehr umfangreich. Sie bieten neben einem Source Code bzw. Ressource Editor sowie einer umfangreichen Sammlung von Klassen und einem detaillierten Hilfesystem auch Debugging Tools sowie einen Windows CE Emulator. Softwareentwickler, die bereits mit Visual Studio vertraut sind, sollten sich auch mit den Embedded Visual Tools schnell zurecht finden.

3.5.4 QNX

POSIX

QNX ist ein 32 Bit Multiuser System und wurde von der kanadischen Firma QNX Software Systems Ltd. entwickelt. Es kann für Echtzeit-Applikationen verwendet werden und unterstützt Multitasking und prioritätsgesteuertes, verdrängendes Scheduling mit schneller Kontextumschaltung. Durch die Orientierung am POSIX Standard fällt Benutzern, die bereits andere UNIX-Derivate, wie Linux, HP-UX, IRIX, Solaris oder SunOS kennen, die Bedienung sehr leicht. Eine flexible Architektur erlaubt sowohl den Einsatz in kleinen eingebetteten Systemen mit minimaler Konfiguration und nur wenigen Kernelmodulen, als auch in großen, im Netzwerk verteilten Anwendungen. Man muss daher nur die Ressourcen installieren, die tatsächlich benötigt werden. Entwickler können QNX auch sehr einfach an die Bedürfnisse ihrer eigenen Applikationen anpassen.

Die Effizienz, Modularität und Einfachheit erreicht QNX durch zwei fundamentale Prinzipien, seine Microkernel-Architektur und die nachrichtenbasierte Kommunikation zwischen Prozessen. QNX besteht aus einem kleinen Kern mit einer Gruppe kooperierender Prozesse (Prozess-Manager, Geräte-Manager, verschiedene Filesystem-Manager, Netzwerk-Manager, die graphische Bedienoberfläche Photon microGUI usw.). Sie ergänzen je nach Bedarf den Microkernel. Alle Module außer dem Prozess-Manager können während der Laufzeit gestartet oder beendet werden (Quelle: www.qnx.com).

Prozess-Manager

Prozess-Manager:
Der Prozess-Manager ist mit dem Microkernel in einem Modul verbunden. Dieses Modul ist für alle Laufzeitsysteme erforderlich. Der Prozess-Manager benutzt den gleichen Adressraum wie der Microkernel. Er wird vom Microkernel wie ein Prozess gehandhabt und nutzt zur Kommunikation mit anderen Prozessen im System das Message Passing (Nachrichtenaustausch) des Microkernels. Der

Prozess-Manager ist für die Erzeugung neuer Prozesse im System verantwortlich und verwaltet außerdem die grundlegenden Prozess-Ressourcen. Diese Dienste werden alle über Nachrichten angeboten. Will z. B. der Prozess-Manager einen neuen Prozess erzeugen, verschickt er die Informationen in Form einer Nachricht. Da Nachrichten netzwerkweit funktionieren, kann sie zum Erzeugen eines neuen Prozesses einfach an den Prozess-Manager des entfernten Netzwerkknotens versandt werden.

Geräte-Manager

Geräte-Manager:
Der Geräte-Manager von QNX bildet die Schnittstelle zwischen Prozessen und zeichenorientierten Terminal-Geräten. Weiterhin reguliert er den Datenfluss zwischen einer Anwendung und dem Gerätetreiber.

Dateisystem-Manager

Dateisystem-Manager:
QNX bietet eine Auswahl an Dateisystemen an, die gleichzeitig nebeneinander laufen können. Wie die meisten Dienste des Betriebssystems, werden sie außerhalb des Kernels ausgeführt. QNX unterteilt sie in fünf Kategorien: Image, Block, Flash, Network und Virtual Filesysteme.

Power-Manager, Power Management Framework

Power-Manager:
Als die zentrale Komponente innerhalb des Power Management Frameworks, ist der Power-Manager verantwortlich für den Energiemodus (engl. power mode) jeder einzelnen vom Framework verwalteten Komponente. Das Framework ermöglicht eine feingranulare Kontrolle der Leistungsaufnahme jeder einzelnen Systemkomponente. Besonders hervorzuheben ist die Eigenschaft des Frameworks, es dem Anwendungsentwickler und nicht etwa dem Betriebssystem zu erlauben, individuelle Strategien für die Leistungsaufnahme (engl. power policies) für jedes System zu definieren. Dies ermöglicht letztendlich schnellere Systemantwortzeiten genauso wie längere Batterielebensdauern.

GUI

Photon microGUI:
Viele eingebettete Systeme benötigen für die Interaktion mit der Applikation ein User Interface (UI). Für komplexe Applikationen oder für maximale Benutzerfreundlichkeit ist eine graphische Benutzeroberfläche (GUI) mit mehreren Fenstern wünschenswert. Graphische Fenstersysteme wie wir sie von Betriebssystemen im Desktop-Bereich her kennen benötigen allerdings zu viele Systemressourcen als dass sie für den Einsatz in eingebetteten Systemen praktikabel wären. Photon geht bei der Erzeugung einer GUI einen

völlig neuen Weg; es benutzt, im Gegensatz zu anderen monolithischen Fenstersystemen, einen Microkernel und eine Menge kooperierender Prozesse. Als Ergebnis liefert Photon eine GUI mit bemerkenswerten Eigenschaften (Quelle: www.qnx.com):

- Bei knappen Speicherressourcen bietet Photon einen hohen Grad an Fensterfunktionalität in Umgebungen, in denen sonst nur eine kleine Grafikbibliothek Platz hätte.
- Photon bietet eine sehr flexible, durch den Benutzer ausbaufähige und skalierbare Architektur, die es Entwicklern ermöglicht, die GUI an ihre eigenen Applikationen anzupassen.
- Durch die flexible Anbindung verschiedener Plattformen können Photon-Applikationen von jeder virtuell verbundenen Bildschirmumgebung genutzt werden.

Programmieren unter QNX

Programmieren unter QNX:
In den Softwarebibliotheken von QNX findet sich u. a. eine komplette GNU-Werkzeugkette. Mit ihrer Hilfe ist Portierung von Linux-Software auf QNX möglich. Besonders hervorzuheben ist auch der Photon Application Builder: Mit seiner Hilfe lassen sich in kurzer Zeit Applikationsprototypen erstellen.

3.5.5 Embedded Linux

POSIX, GPL

Linux ist ein Multiuser Multitasking 32 Bit und 64 Bit POSIX konformes Betriebssystem, welches hauptsächlich in Internetservern und in den letzten Jahren auch zunehmend auf Desktops eingesetzt wird. Unter Embedded Linux versteht man eine Zusammenstellung von Linux Kernel und Software, die speziell auf eingebettete Systeme zugeschnitten ist, sich grundsätzlich aber nicht von dem Linux, das auf Servern und Desktops eingesetzt wird, unterscheidet. Genau genommen bezieht sich die Bezeichnung *Linux* lediglich auf den Kernel des Betriebssystems. Werkzeuge und Software, die auf diesem Kernel aufbauen, kommen aus dem *GNU Projekt* und fallen, ebenso wie der Kernel, unter die GNU Public License (GPL). Sie sind nicht Linux spezifisch, sondern können in den meisten Fällen auch auf anderen Unix-artigen, teilweise sogar Windows Betriebssystemen eingesetzt werden.

Linux ist multitaskingfähig. Standard-Distributionen sind jedoch nicht für den Echtzeitbetrieb geeignet. Hierfür wurden insbesondere zwei spezielle Linux-Systeme implementiert, RT-Linux und RTAI

(Real Time Application Interface). Diese Systeme verfügen über einen echtzeitfähigen Kernel.

3.6 Zusammenfassung

Unter der Echtzeitfähigkeit eines Betriebssystems versteht man in erster Linie dessen reale Fähigkeit, in einer gegebenen Betriebsumgebung alle anstehenden Aufgaben und Funktionen unter allen Betriebszuständen immer rechtzeitig und ohne Ausnahme erledigen zu können. „Rechtzeitig“ oder „in Echtzeit“ versteht sich somit nicht als exakte wissenschaftliche Definition, sondern als sehr variable Größe, die sich nach den jeweiligen (Echtzeit-) Anforderungen der spezifischen Anwendungen und deren zeitlichen Rahmenbedingungen orientiert und ausrichtet. Ein Echtzeitsystem ist also ein eingebettetes System, das Echtzeitanforderungen besitzt und dann ggf. mit Hilfe eines Echtzeitbetriebssystems implementiert werden kann.

Ein Echtzeitbetriebssystem führt zwischen der Hardware und den darauf auszuführenden Applikationen eine Abstraktionsebene ein und ermöglicht es Softwareentwicklern auf diese Weise, von der Plattform unabhängige echtzeitfähige Applikationen zu programmieren, welche die geforderten Echtzeitbedingungen einzuhalten versuchen. Das zeitliche Verhalten eines Echtzeitsystems wird so vorhersagbar bzw. deterministisch. Um dies zu unterstützen, enthalten Echtzeitbetriebssysteme spezielle Scheduling Algorithmen für die Einplanung der zeitlichen Reihenfolge der in der Applikation enthaltenen Teilaufgaben. VxWorks von der Firma Windriver Systems ist ein speziell für Steuerungs- und Datenerfassungszwecke entwickeltes Echtzeitbetriebssystem. Es wird im Flugzeugbau genauso verwendet wie im Automobilbau bzw. im Motorsport. Als Open Source Echtzeitbetriebssystem steht beispielsweise eCOS zur Verfügung.