Kurseinheit 13

Echtzeitbetriebssysteme

Lernziele

Nach dem Durcharbeiten dieser Kurseinheit sollten Sie

- die speziellen Anforderungen benennen können, die hinsichtlich verschiedener Kriterien an ein Echtzeitbetriebssystem gestellt werden,
- einen Eindruck von den aktuellen technischen Entwicklungen haben,
- einige Konsequenzen wirtschaftlicher Überlegungen bezüglich Programmentwicklung und -wartung kennen,
- in der Lage sein, die grundlegende Struktur eines portierbaren Echtzeitbetriebssystems zu skizzieren sowie Vorteile der Portierbarkeit anzugeben,
- verschiedene Strategien zur Vergabe von Prozessorzeit an Tasks beschreiben können,
- wissen, in welchen Fällen diese Strategien zeitgerecht sind,
- die Ziele und Probleme der Zuteilbarkeitsanalyse kennen,
- wissen, welche Schwierigkeiten bei der Erreichung deterministischen Zeitverhaltens auftreten können, sowie
- wichtige Fachbegriffe des Themengebiets „Prozessorzuteilung“ kennen.

13.1 Anwenderorientierte Erwägungen

13.1.1 Allgemeine Anforderungen an Echtzeitbetriebssysteme

Echtzeitbetriebssysteme unterscheiden sich von entwicklungsorientierten Betriebssystemen im wesentlichen durch die Forderung nach präzisem Zeitverhalten. Echtzeitsysteme sind überwiegend eingebettete Systeme, die sehr spezielle Anforderungen an Echtzeitbetriebssysteme stellen. Darum genügt es nicht, schlicht Echtzeitbetriebssysteme zu betrachten und diese gegenüber entwicklungsorientierten Betriebssystemen abzugrenzen. Vielmehr müssen die spezifischen Anforderungen an Echtzeitbetriebssyteme berücksichtigt werden. Das sind unter anderem:

- präzises Zeitverhalten mit kurzen Antwortzeiten,
- Verkehr mit sehr speziellen Peripheriegeräten,
- effiziente Speichernutzung mit der Möglichkeit, ohne Externspeicher auszukommen,
- Fähigkeit zur Bildung kleinster Systeme, die mit einem Minimum an Strom und ohne Bedienung auskommen müssen,
- Herstellerneutralität und gute Portierbarkeit zur Sicherung langfristiger Systemnutzung, denn eingebettete Systeme haben für Rechnersysteme ungewöhnlich lange Lebensdauern von bis zu mehreren Jahrzehnten und überdauern mehrere Generationen von Mikroprozessoren,
- sehr hohe Betriebssicherheit, die auch die Überwachung der Hardware bedingt.

Die besonderen Anforderungen an Echtzeitbetriebssysteme können kaum erfüllt werden, wenn sie nicht bereits beim Entwurf berücksichtigt wurden. In der Praxis werden sie auch nur selten erfüllt.

Die im Entwurf steckenden – oder fehlenden – Ideen bestimmen die Güte von Software. Fehler in Software, die schon durch schlechte Entwürfe entstehen, lassen sich später auch durch noch so intensives Testen nicht mehr aufdecken, weil in Bezug zur Spezifikation getestet wird, und bleiben als latente Ursachen zukünftigen Versagens erhalten. Die Qualität von Software läßt sich nicht durch physikalische Testverfahren prüfen, sondern nur nach Kriterien wie Robustheit, Einfachheit des Entwurfs oder Anpassungsfähigkeit an wechselnde Anforderungen beurteilen. Diese Schwierigkeit oder gar Unmöglichkeit einer Qualitätsprüfung beeinflußt entscheidend die (dreiste) Art üblicher Werbeaussagen. Es ist nicht sinnvoll, einzelne Leistungsaussagen – wie z.B. über Prozeßwechselzeiten – kritiklos miteinander zu vergleichen. Schließlich besteht die einfachste, am ehesten Erfolg versprechende – und darum am häufigsten genutzte – Methode, für günstige „Zahlen“ zu sorgen, genau darin, ganz gezielt spezielle vorteilhafte Geräteeigenschaften hervorzuheben.

Betrachtet man die hohen Entwicklungskosten für ein zuverlässiges Programmsystem einerseits und die Turbulenzen auf dem Hardware-Sektor andererseits, so verbietet es sich von selbst, das System auf eine spezielle Hardware-Architektur zu stützen. Software-Entwicklungen – insbesondere komplexe Systeme – haben lange Zeit unter den Einflüssen der schnellebigen Hardware gelitten. Dies gilt vor allem für langlebige Systeme. Selbst komplexe Software-Produkte wie Echtzeitbetriebssysteme werden erst allmählich zu Produkten geräteunabhängiger Hersteller. Es gilt, größte Aufmerksamkeit auf eine gute Portierbarkeit von Software-Systemen zu richten. Voraussetzungen für leichte Portierbarkeit von Echtzeitbetriebssystemen sind

- Konfigurierbarkeit von Geräteparametern, soweit dies möglich ist, sowie
- Definition hinreichend gerätenaher Programmschnittstellen, unter der alle gerätespezifischen Eigenschaften formuliert werden können und auf die sich die Systeme konsequent stützen.

Eingebettete Systeme haben in der Regel eine sehr spezielle Peripherie. Auch dieser Umstand verlangt gute Konfigurierbarkeit der Software-Komponenten und ausgeklügelte, gut dokumentierte Treiberschnittstellen. Beides ist in der Praxis nur selten gegeben – insbesondere bei Betriebssystemen, die auf bestimmte Hardware zugeschnitten sind. Noch seltener geht die Konfigurierbarkeit der Dienste eines Echtzeitbetriebssystems so weit, daß sie auch noch Minimalsysteme zuließe, obwohl die immer noch stetig sinkenden Hardware-Kosten inzwischen deutlich für den Einsatz entsprechend reduzierter Betriebssysteme in derartigen Systemen sprechen. Die bisher hier übliche Assembler-Programmierung direkt auf der Hardware ist in keinem Fall mehr wirtschaftlich gerechtfertigt.

Echtzeitbetriebssysteme, deren Ablaufsteuerung lediglich mit statischen Prioritäten arbeitet, führen in der Praxis zu erheblichen Problemen beim Entwurf größerer Anwendungssysteme. Der Entwickler kann nur über die Vergabe statischer Prioritäten Einfluß auf die zeitliche Abfolge von Tasks nehmen. Genauer betrachtet kann er dies nur beim Systementwurf selbst tun. Er weiß folglich nicht, welche Tasks zu einem konkreten Zeitpunkt ablaufbereit sind und somit um die Prozessorzuteilung konkurrieren.

Zeitgerechte Reaktionen von Tasks (Antwortzeiten) können damit nicht garantiert werden. Ein antwortzeitgesteuertes Echtzeitbetriebssystem ist in der Lage, den Ablauf rechenwilliger Tasks zu jedem Zeitpunkt so zu steuern, daß sämtliche Antwortzeiten eingehalten werden, falls die verfügbare Rechenleistung dies überhaupt erlaubt. Im Gegensatz zu Prioritäten ergeben sich Antwortzeiten ganz von selbst aus der Natur der externen Ereignisse heraus, auf die jeweils zu reagieren ist.

Die mit statischen Prioritäten arbeitenden Zuteilungsalgorithmen sind insbesondere nicht „fair“, d.h. bei hochbelastetem Prozessor berücksichtigen sie Tasks mit niedriger Priorität überhaupt nicht mehr. Diese Erkenntnis hat zu sehr kuriosen und in der Regel wenig wirksamen Algorithmen geführt. Ein Beispiel:

> Ein Prioritätenalgorithmus beinhaltet eine statische, eine dynamische und eine Ereignispriorität. Ein Prozeß besitzt als Basispriorität seine statische Priorität und kann sich damit von anderen Prozessen unterscheiden. Während seiner Wartezeit auf Zuteilung wächst diese Priorität. Bei Auftreten eines Ereignisses erhält der zugewiesene Prozeß einmalig einen Ereignisbonus.

Eine wichtige Aufgabe von Echtzeitbetriebssystemen für eingebettete Systeme, die häufig vernachlässigt wird, besteht darin, Fehler umfassend zu behandeln. Erkennung und entsprechende Behandlung von Fehlern an Programmschnittstellen sind selbstverständliche Aufgaben, die keiner besonderen Erwähnung bedürfen. Sie werden häufig auch befriedigend gelöst. Weniger üblich, aber im Betrieb unbedingt erforderlich, ist die Überwachung ordnungsgemäßer Gerätefunktion. Hardware kann durch ein BITE-Programm (Built-In Test Equipment) periodisch oder bei Verdacht auf einen (vom Gerätetreiber erkannten) Fehler überprüft werden. Aufgabe des Betriebssystems ist es, eine Kommunikationsschnittstelle zum BITE bereitzustellen. Ziel ist die Führung eines Gerätestatus, der dem Anwendungssystem bei Bedarf zur Verfügung gestellt werden kann. Sinnvolle Reaktionen eines Anwendungssystems erfordern in jedem Fall, daß umfassende Statusinformationen von E/A-Geräten vorhanden sind. An dieser Aufgabe müssen Betriebssystem und Gerätetreiber beteiligt werden.

Die oben aufgeführten und noch recht allgemein gehaltenen Anforderungen sollen im folgenden Abschnitt konkretisiert werden. Der darin aufgestellte Katalog von Forderungen und Gestaltungsvorschlägen an Echtzeitbetriebssysteme beschränkt sich jedoch nur auf die Aspekte, die mit ihrem wirtschaftlichen Einsatz in Zusammenhang stehen. Danach werden wir kurz technische Trends diskutieren. Es schließen sich Betrachtungen zur Wirtschaftlichkeit der Entwicklung und Wartung von Software an, die unter Echtzeitbetriebssystemen ablaufen soll.

13.1.2 Gestaltungsrichtlinien für und spezielle Forderungen an Echtzeitbetriebssysteme

Zunächst wird das Anforderungsprofil für Einprozessorsysteme bzw. Knotenrechner in verteilten Systemen mit einem Prozessor dargestellt.

13.1.2.1 Anforderungen an den Leistungsumfang

- Ein Betriebssystem soll so sicher und fehlertolerant wie möglich sein.
- Ein Betriebssystem muß die Sprachmittel der wesentlichen Echtzeitprogrammiersprachen unterstützen, so daß diese ohne Einschränkungen ihres Sprachumfangs implementiert werden können, d.h. Sprachschalen für diese Sprachen müssen vorhanden oder einfach zu entwickeln sein.
- Zuteilungsalgorithmen sollen auch antwortzeitgesteuert sein, d.h. sie sollen stets derjenigen der ablaufbereiten Tasks den Prozessor zuteilen, welche die kürzeste Antwortzeit besitzt.
- Es muß eine geräteunabhängige Programmierschnittstelle vorhanden sein, damit Anwenderprogramme leicht auf andere Hardware übertragen werden können.
- Handhabung und Installation müssen einfach sein, weil die Anwender an ihren Problemen, jedoch nicht am Betriebssystem interessiert sind (Forderung nach „Unauffälligkeit des Betriebssystems“).
- Ein Betriebssystem muß auch für gängige und preiswerte Hardware-Plattformen wie z.B. Industrie-PCs und Einplatinenrechner zur Verfügung stehen.
- Verschiedene Prozessorfamilien müssen unterstützt werden, um die Anwender nicht in der Auswahl der geeignetsten Hardware einzuschränken.
- Standardwerkzeuge müssen als Tasks aufrufbar sein, um kostspielige Neuentwicklungen zu vermeiden.
- Standardwerkzeuge müssen in der Projektentwicklungsphase eingesetzt werden können, um späteres Umlernen zu vermeiden.
- Ein Betriebssystem muß ein von der Hardware unabhängiges BITE unterstützen.
- Interne Kommunikation und Synchronisation sind für Echtzeitsysteme sehr wichtig und müssen flexibel gehandhabt werden. Wegen der geforderten Vielfalt müssen Betriebssysteme möglichst universell anwendbare Basisoperationen zur Verfügung stellen. Komplexe und hochspezialisierte Operationen (wie etwa das in der Sprache Ada verwendete „Rendezvous“) müssen aus diesen Basisoperationen zusammensetzbar sein.

- Wegen der Vielfalt „genormter“ externer Übertragungsprotokolle und dem ständigen Wandel dieser „Normen“ kann an dieser Stelle keine allgemein gültige Anforderung formuliert werden. Solange sich diese Situation nicht grundlegend ändert, müssen deswegen von Fall zu Fall immer wieder neue Treiber entwickelt werden. Diese sind dann an interne Übertragungskanäle derart anzukoppeln, daß die Anwenderprogramme nur die einheitliche Schnittstelle der internen Kommunikation sehen.
- Betriebssysteme soll Anwendungsprogrammierern die Möglichkeit bieten, eigene Programme zur Behandlung bestimmter Ausnahmen und Signale anzuschließen.
- Betriebssysteme müssen verteilte Systeme unterstützen.
- Betriebssysteme müssen von bestimmter Hardware und von Geräteherstellern unabhängig sein.
- Betriebssysteme müssen die Unabhängigkeit der Anwendungsprogramme von der Geräteentwicklung gewährleisten.
- Betriebssysteme müssen gemäß der Richtlinie VDI/VDE 3554 [5] aus Gründen der Portierbarkeit, Klarheit und Komplexitätsbeherrschung in Schichten gegliedert sein. Dies impliziert u.a. auch, daß jede Schicht vom Benutzer erweitert werden kann.
- Treiber für spezielle Hardware müssen leicht in Betriebssysteme integriert werden können.

13.1.2.2 Architektur und Konfigurierbarkeit

- Der Kern eines Betriebssystems muß speicherresident, d.h. ohne Massenspeicher, und für sich allein betreibbar sein.
- Ein Betriebssystem muß ROM-fähig sein, d.h. es darf kein Massenspeicher notwendig sein, um es zu laden. Als positiver Nebeneffekt wird die Sicherheit durch die strikte Trennung von Code und Daten erhöht.
- Es muß möglich sein, ein System zu generieren, bei dem Code und Daten von Betriebs- und Anwenderprogrammen ausschließlich im Hauptspeicher liegen (System mit fester Hauptspeicherzuweisung).
- Durch direkte Übersetzung der Quellen soll für jede dedizierte Anwendung die minimale Betriebssystemkonfiguration erzeugbar sein.
- Betriebssysteme müssen ohne Dateiverwaltung betrieben werden können.
- Betriebssysteme müssen ohne Bedienmonitor betrieben werden können.
- Betriebssysteme müssen portierbar sein, d.h. sie sollen in weitverbreiteten Sprachen programmiert sein, und alle rechnerspezifischen Teile müssen durch wohldefinierte Schnittstellen von den übrigen Teilen getrennt sein.
- Anwenderprogramme und Betriebssysteme sollen voneinander unabhängig gebunden und geladen werden können. Damit ist es möglich, Anwenderprogramme für RAMs zu entwickeln, während das Betriebssystem im ROM steht. Das verkürzt die Programmladezeiten und erspart langwieriges Brennen von PROMs. (Das Verfahren erfordert eine zumindest eingeschränkte Konfigurierbarkeit von Firmware-Betriebssystemen.)
- Der Code von Systemkernen soll nicht länger als einige Dutzend KBytes sein. Damit kann in etwa sichergestellt werden, daß Kerne effizient programmiert sind.

- Betriebssysteme sollen auf marktgängigen Rechnern – wenn möglich auf PCs und/oder Workstations – zu generieren sein.

13.1.2.3 Dokumentation

- Es müssen Systemhandbücher folgenden Inhalts vorhanden sein:
  - Beschreibung aller Schnittstellen zu den Anwenderprogrammen,
  - Beschreibung der Architektur und der externen Schnittstellen der Betriebssystemkerne,
  - Beschreibung der Treiberschnittstellen,
  - Anleitung zur Systemgenerierung.

  Diese Forderung macht Betriebssysteme „beistellfähig“, d.h. sie können ohne große Hilfe von Systementwicklern eingesetzt und auch angepaßt werden.
- Der Quellcode der Betriebssystemkerne muß verfügbar sein. Dies ist für die Suche nach schwierig zu findenden Programmfehlern unerläßlich.
- Es sollen alle Schnittstellen zwischen den maschinenabhängigen und den übrigen Systemteilen beschrieben sein, um Portierbarkeit sicherzustellen.
- Zur Entwicklung von Übersetzern soll genau erläutert sein, auf welche Weise sprachabhängige Laufzeitsysteme an Betriebssystemkerne anzupassen sind.
- Zur Programmpflege sollen Quellcode, Architektur- und Schnittstellenbeschreibungen aller Teile von Betriebssystemen verfügbar sein.

13.1.2.4 Vertragsrechtliche Anforderungen

- Für Betriebssysteme müssen uneingeschränkte Nutzungsrechte zu erwerben sein.
- Die obigen Forderungen zur Dokumentation müssen sich durchsetzen lassen.
- Nachbaurechte müssen zu erwerben sein.

13.1.2.5 Anforderungen an die Dateiverwaltung

- Die Dateiverwaltung von Betriebssystemen soll kompakte Dateien für direkten Zugriff mit festgesetzter Datei- und Satzlänge unterstützen. Auf diese Dateien soll auch byteweise zugegriffen werden können.
- Die Dateiverwaltung soll segmentierte Dateien für sequentiellen Zugriff mit variabler Datei- und Satzlänge unterstützen. Diese Dateien sollen sich beim Schreiben über ihr aktuelles Ende hinaus automatisch segmentweise verlängern. Die Segmentlänge soll bei der Systemgenerierung festgelegt werden können.
- Das Dateiformat muß weitverbreitet sein, damit erfaßte Meßdaten mit preiswerten Standardprogrammen weiterverarbeitet werden können.

13.1.2.6 Anforderungen an die Ein- und Ausgabe

Weil Echtzeitsysteme viel Spezialperipherie aufweisen, können standardisierte Betriebssysteme hier nicht viel Unterstützung bieten. Die Anforderungen an die Ein- und Ausgabe konzentrieren sich daher auf das Allernötigste: Datensichtgeräte, Drucker, Externspeicher und Dateien.

13.1.2.7 Anforderungen an die Fehlerbehandlung

Betriebssysteme können nicht vor Fehlbedienung oder Eingabe falscher Daten schützen. Dies ist Aufgabe der Anwenderprogramme. Betriebssysteme sollen lediglich sichere Umgebungen für die Anwenderprogramme bieten. Das bedeutet vor allen Dingen zweierlei:

1. Ein System soll gewisse Regeln beim Ablauf von Programmen überwachen und Verstöße gegen diese Regeln sofort melden.
2. Die Auswirkungen solcher Verstöße sollen auf das absolute Minimum beschränkt werden.

Eine typische Regel dieser Art lautet:

> „Keine Task darf ein Betriebsmittel freigeben, das ihr nicht zugewiesen wurde."

Eine typische Maßnahme, die Auswirkungen von Fehlern zu beschränken, besteht darin, daß fehlerhafte Tasks zwangsweise beendet werden.

Es sind u.a. folgende Ablaufregeln zu überwachen:

- Betriebssysteme müssen die richtige Reihenfolge von Reservierung und Freigabe der Betriebsmittel überwachen.
- Betriebssysteme sollen Schnittstellen für alle von Anwenderprogrammen aufgedeckten Ausnahmesituationen besitzen.
- Betriebssysteme sollen die Blockade des Prozessors durch eine in eine Endlosschleife geratene Task entdecken.
- Betriebssysteme sollen die Einhaltung etwaiger Budgets für bestimmte Betriebsmittel überwachen.
- Betriebssysteme sollen bei Zugriffen auf Betriebsmittel prüfen, ob die zugreifenden Tasks die dazu nötigen Privilegien besitzen.
- Betriebssysteme sollen nicht realisierbare Einplanungen auf Fortsetzung angehaltener Tasks erkennen.
- Betriebssysteme sollen die Maximalzahl der wirksamen Einplanungen zur Aktivierung einzelner Tasks überwachen.

13.1.2.8 Anforderungen an die Überwachungsfunktionen

Monitore sind stark von der Hardware abhängige Betriebssystemteile und daher nur sehr bedingt portierbar. Bei Systemen ohne Programmladegeräte, deren Programme aber trotzdem im RAM gespeichert werden sollen, dienen Monitore zum Laden der Programme von Wirtssystemen aus. Außerdem sind Monitore sehr nützliche Hilfsmittel bei Integration und

Test von Software. Sehr oft sind sie während der Nutzungsphase die einzigen Werkzeuge für die Fehlersuche vor Ort.

- Monitore müssen konfigurierbar sein. Einzelne Funktionen, insbesondere Disassembler, müssen „wegkonfigurierbar“ sein.
- Monitore müssen sich in zwei Teile – Oberteil und Unterteil – trennen lassen. Oberteile müssen auf als Wirtssysteme fungierende Anlagen auslagerbar sein, auf denen auch ganze Betriebssysteme generiert werden können.
- Die in Betriebssysteme eingebundenen Unterteile und die in Wirtssysteme ausgelagerten Oberteile müssen über Datenverbindungen kommunizieren können.
- Monitorunterteile sollen möglichst wenig Speicher benötigen.
- Die Kommunikation zwischen Ober- und Unterteil muß im Unterteil über eine interne Schnittstelle laufen, die von der Unterbringung des Oberteils und ggf. auch von der Art der Datenverbindung zwischen beiden Teilen unabhängig ist. Die Schnittstelle sowie die Formatierung der Daten und Programme müssen klar dokumentiert sein. Diese Forderung sichert die Austauschbarkeit von Monitoroberteilen (ggf. samt Wirtssystemen) ohne wesentliche Änderungen an den Unterteilen.

13.1.2.9 Zusätzliche Anforderungen bei Mehrprozessorsystemen

- Betriebssysteme müssen homogene Multiprozessorsysteme mit enger, loser und gemischter Kopplung unterstützen.
- Betriebssysteme müssen innerhalb größerer Gesamtsysteme einzelne Untersysteme aus speichergekoppelten Prozessoren unterstützen.
- Betriebssysteme müssen die Tatsache, daß sich Untersysteme aus getrennten Einzelprozessoren zusammensetzen, soweit wie möglich vor den Anwenderprogrammen verbergen.
- Betriebssysteme müssen innerhalb von Untersystemen das Konzept der Schattenprozessoren unterstützen, d.h. von Prozessoren, die ohne Wirkung nach außen zur Sicherheit parallel mitarbeiten.
- Betriebssysteme sollen innerhalb von Untersystemen Prozessorenpaare aus Haupt- und Ersatzprozessor sowie Prozessorengruppen unterstützen.
- Betriebssysteme müssen einheitliche Zeitnormale aufrechterhalten.
- Betriebssysteme müssen in der Lage sein, mehrere alternativ aktivierbare Treiber für ein Externgerät zu unterstützen.

13.1.3 Technische Entwicklungen

Im Hinblick auf die ständig steigende Rechenleistung der Mikroprozessoren in Verbindung mit immer besserer Speicherausstattung nimmt der Zwang zu zeit- und speichereffektiver (Assembler-) Programmierung ab. An die erste Stelle treten statt dessen erhöhte Anforderungen an die Programmsicherheit.

Programme sollen alle sicherheitstechnischen Prozessormerkmale nutzen. Sie sollen robuster werden, nicht nur gegenüber fehlerhafter Bedienung, sondern auch gegenüber der sich ständig weiterentwickelnden Hardware. Durch geeignete Abgrenzung sollen sie sich den

notwendigen Freiraum für ihre Entwicklung und vor allem solche Rahmenbedingungen schaffen, die auch längerfristige Investitionen rentabel erscheinen lassen. Gute Software bekommt damit die Chance, ein eigenständiges Produkt von hoher Qualität zu werden.

Neben der Gestaltung der Software spielen aber auch die Einrichtungen der Hardware eine ganz große Rolle. Nicht der schnellste Prozessor ist der beste, sondern der sicherste. Geschwindigkeit wird durch den Einsatz parallel arbeitender Prozessoren gewonnen. Echtzeitbetriebssysteme sind erst allmählich dabei, auf diesem Feld einiges aufzuholen. Multiprozessorsysteme werden nicht nur aus Gründen der Lastverteilung, sondern auch aus sicherheitstechnischen Überlegungen heraus größere Bedeutung erlangen.

Eine Entwicklung hin zur Standardisierung eines oder mehrerer Echtzeitbetriebssysteme selbst, so wie wir sie von der nicht zeitkritischen Datenverarbeitung her kennen, ist nirgends auszumachen. Im Jahre 1984 wurde in Japan das Projekt TRON („The Real-Time Operating System Nucleus“) mit dem Ziel initiiert, einen umfassenden Architekturstandard über eine Familie kompatibler Betriebssysteme global zu verbreiten. Der Architekturstandard sollte alle Anwendungsgebiete und Größenordnungen von Rechnern abdecken. Ein Mitglied dieser Betriebssystemfamilie („ITRON“) zielte auf industrielle Echtzeitanwendungen ab und wäre mithin ein Kandidat für einen internationalen Standard auf dem Gebiet der Echtzeitbetriebssysteme gewesen. Da TRON bisher jedoch außerhalb von Japan kaum von sich reden gemacht hat, bleibt die Entwicklung abzuwarten. Ein Erfolg dieser Initiative ist in absehbarer Zeit höchst unwahrscheinlich.

Auf dem Gerätesektor schreitet die Entwicklung zwar schnell fort, jedoch ohne daß sich grundsätzlich etwas an der seit den 1940er Jahren eingeführten von Neumann-Architektur ändert. Auch die im Rahmen des TRON-Projektes neu entwickelten Prozessoren basieren auf diesem klassischen Konzept, und RISC-Prozessoren sind den Ursprüngen näher als moderne CISC-Prozessoren. Da also Sprünge hinsichtlich wesentlich veränderter Hardware-Architekturen in absehbarer Zeit nicht zu erwarten sind, müssen Echtzeitbetriebssysteme auch weiterhin das von Neumann-Konzept unterstützen, jedoch auf einer Ebene, die von den Eigenheiten spezieller Prozessoren abstrahiert. Sie sollten also in relativ einfachen, aber dennoch höheren, imperativen Programmiersprachen formuliert sein. Dadurch ergibt sich eine leichte Portierbarkeit auf neu am Markt erscheinende Prozessoren. Weil es leider auch noch keine standardisierten Hardware-Architekturen gibt, muß in Kauf genommen werden, daß im Rahmen jeder Portierung ein Rest von Anpassungen auf dem Assembler-Niveau nicht zu vermeiden ist.

13.1.4 Wirtschaftlichkeit der Programmentwicklung

Die Kosten der Entwicklung von Programmsystemen sind beträchtlich. Dies gilt besonders dann, wenn wie im Falle von Echtzeitbetriebssystemen hohe Qualitätsanforderungen gestellt werden. Betrachtet man ferner, daß Programmentwicklungskosten regelmäßig unterschätzt werden, so liegt es nahe, bei der Durchführung eines hypothetischen Projektes „Entwicklung eines Echtzeitbetriebssystems“ zunächst die Qualitätsanforderungen zu reduzieren. Man könnte glauben, daß dies anfangs ohne weiteres zu akzeptieren wäre, da ja in weiteren Projektphasen entsprechende Verbesserungen nachgeholt werden könnten. Das ist jedoch ein Trugschluß, dem bereits „Generationen“ von Programmentwicklern aufgesessen sind. Es gilt ganz klar festzuhalten, daß die Ideen, die in einem Programmentwurf stecken oder fehlen, auch die Güte der Software bestimmen. Die Fehler, die ein schlechter Entwurf in eine Software einbringt, lassen sich nachher auch durch noch so intensives Testen nicht mehr aufdecken. Schlechte Software enthält Tausende latenter Fehlermöglichkeiten, von denen jedoch nur ein Bruchteil ans Tageslicht tritt. Als Konsequenz aus diesen Er-

fahrungen predigen manche Programmentwicklungsleiter schon seit langer Zeit, man möge doch mehr Zeit und Verstand auf die Entwurfsphase von Programmierprojekten verwenden. Leider sprechen oft sehr kurzfristige Termine und entsprechend kurzsichtige materielle Interessen dagegen.

Echtzeitbetriebssysteme entstehen nicht von heute auf morgen. Sie haben eine lange Lebensdauer (verdient). Der Sicherung der langen Verwendbarkeit eines Echtzeitbetriebssystems ist demnach besonderes Augenmerk zu schenken. Lange Verwendbarkeit ist – betrachtet man die wohl unvermeidlichen Hardware-Turbulenzen – gleichzusetzen mit guter Portierbarkeit, und die wiederum ist eine sehr wünschenswerte Eigenschaft auch dann, wenn man nur die Gegenwart betrachtet. Schließlich stellt sie sicher, daß ein Produkt in einem breiten Spektrum von Hardware-Plattformen eingesetzt werden kann. Daraus folgt das Fazit:

> Gute Portierbarkeit ist als das am höchsten zu bewertende Wirtschaftlichkeitsmerkmal von Echtzeitbetriebssystemen anzusehen.

Leider bedeutet die Sicherung guter Portierbarkeit erheblich erhöhte Investitionen in die Entwurfsphase und fällt deshalb in der Praxis oft kurzfristigen Interessen zum Opfer. Ganz unglücklich wirkt sich letztlich aus, daß immer noch viel Geld mit dem Verkauf von Hardware verdient wird, daß also Auch-Gerätehersteller am ehesten Geld für die Entwicklung eines Echtzeitbetriebssystems erübrigen, daß sie dieses Betriebssystem natürlich mit ihrer Hardware eingesetzt sehen wollen und daß – leider – der Verzicht auf gute Portierbarkeit auch noch sehr leicht zu guten Verkaufsargumenten verhilft. Der Verzicht auf Portierbarkeit ermöglicht es nämlich, gezielt auf spezielle Geräteeigenschaften einzugehen, was natürlich im Vergleich mit einem herstellerneutralen, portablen Betriebssystem an so mancher Stelle zunächst erstaunliche Leistungswerte liefert.

Im Gegensatz zur Frühzeit der Datenverarbeitung findet die Entwicklung von Anwenderprogrammen heute nicht mehr notwendigerweise auch auf dem Zielsystem statt. Das gilt insbesondere für eingebettete Echtzeitsysteme. Deshalb kann ein Echtzeitbetriebssystem völlig unabhängig von einer möglichst komfortablen Entwicklungsumgebung ausgewählt werden, die man heute üblicherweise in Form einer UNIX-Workstation oder eines PCs finden wird. Es ist als Qualitätsmerkmal einzuschätzen, wenn ein Echtzeitbetriebssystem die Programmentwicklung unter diesen Bedingungen zuläßt und nicht dazu zwingt, selbst dabei angewendet zu werden.

Für Anwender und Entwickler von Echtzeitsystemen wäre es natürlich am wirtschaftlichsten, wenn es ein standardisiertes Echtzeitbetriebssystem gäbe, das zusammen mit auf ihm basierenden Werkzeugen von mehreren konkurrierenden Herstellern bezogen werden könnte. Leider ist das (noch) nicht der Fall. Darum ist mit der Entscheidung für den Einsatz eines Echtzeitbetriebssystems auch immer die Entscheidung für ein ganz bestimmtes System sowie für seinen Hersteller verbunden, zu dem man sich damit in ein gewisses Abhängigkeitsverhältnis begibt. Neben technischer Kompetenz des Herstellers oder Preis des Produktes ist diesem Abhängigkeitsverhältnis bei der Entscheidungsfindung besondere Beachtung zu schenken. So ist darauf zu achten, daß alle Preise transparent sind und daß man wirklich alle Nutzungsrechte und Dokumentationen des Echtzeitbetriebssystems erwirbt, um vor unerwarteten und unliebsamen Nachforderungen sicher zu sein. Weiterhin ist die Datenverarbeitungsbranche stark in Bewegung, wodurch auch große Unternehmen mit Milliardenumsätzen in Mitleidenschaft gezogen werden können. Es kann also die Existenz eines Herstellers während der – insbesondere bei eingebetteten Systemen recht langen – Nutzungsdauer einer Anwendung nicht garantiert werden. Darum ist es für den wirtschaftlichen Einsatz eines Echtzeitbetriebssystems unabdingbar, bei seinem Kauf vom

Hersteller die vollständigen Quellprogramme, umfassende Dokumentationen und eine intensive Ausbildung der eigenen Entwicklungsgruppe zu erhalten. So macht man sich vom Verhalten des Herstellers und seinem Schicksal im Wettbewerb unabhängig. Solange der Hersteller existiert und sein Produkt unterstützt, kann man natürlich von seinen Diensten Gebrauch machen, was i.a. kostengünstiger sein dürfte als die Unterhaltung einer eigenen (größeren) Abteilung. Dann bedarf es nur einiger weniger Spezialisten für das Betriebssystem im eigenen Hause, die mit Unterstützungsaufgaben für die Anwendungsentwickler und dem Schreiben von Treiberroutinen für Spezialperipherie voll ausgelastet sein dürften. Diese Personen sind jederzeit in der Lage, eine Abteilung zur Pflege des Betriebssystems aufzubauen, zu schulen und anzuleiten, sobald die Unterstützung durch den Hersteller aus irgendeinem Grunde aufhört. Ihre mit den Jahren zunehmende Vertrautheit mit dem Echtzeitbetriebssystem wird sich sehr günstig auf die Dauer neuer Anwendungsprojekte und auf die Qualität der dabei entwickelten Produkte auswirken. Es soll noch einmal ganz dringend darauf hingewiesen werden, daß leichte Portabilität eines gewählten Echtzeitbetriebssystems unabdingbare Voraussetzung dafür ist, um auch unter ungünstigen, den Hersteller betreffenden Umständen die Investitionen in selbst entwickelte Anwendungsprogramme während ihrer jahrzehntelangen Nutzungsdauer zu sichern, ohne dabei auf den Einsatz neuer und leistungsfähigerer Hardware verzichten zu müssen.

Da es keine Standards, sondern nur proprietäre Echtzeitbetriebssysteme von Geräteherstellern und unabhängigen Software-Häusern gibt, liegt der Gedanke nahe, nicht nur Anwenderprogramme, sondern auch das Betriebssystem selbst zu entwickeln. Auf den ersten Blick scheint diese Lösung bestechend günstig und uneingeschränkt vorteilhaft zu sein. Man kann erwarten, daß keine Anwenderwünsche unerfüllt bleiben müssen, da alle Komponenten des Betriebssystems jederzeit zugänglich und leicht modifizierbar sind. Dieses Vorgehen birgt jedoch große Gefahren in sich, vor denen eindringlich gewarnt werden muß. Die einfache Zugriffsmöglichkeit auf die Entwicklungsgruppe führt dazu, daß Anwenderwünsche allzuleicht in Systemanforderungen umgesetzt werden, darunter auch temporäre, wenig durchdachte Anforderungen. So wird das Echtzeitbetriebssystem entweder auf Kosten der Leistung und der Wartbarkeit zu schwerfällig werden, oder es entsteht schnell eine nicht mehr zu überblickende Fülle verschiedener Versionen. Von der Eigenentwicklung eines Echtzeitbetriebssystems ist aber auch noch aus einer Reihe anderer Gründe abzuraten. So nähme sie einen beträchtlichen Zeitraum in Anspruch, an dessen Ende ein dann noch nicht voll ausgereiftes und nicht praxisbewährtes Produkt stünde, dazu von geringerer Qualität als die bereits auf dem Markt verfügbaren Echtzeitbetriebssysteme. Das liegt daran, daß die für eine solche Entwicklung notwendigen hochqualifizierten Spezialisten kaum in ausreichender Zahl zu gewinnen sind. Schätzt man den Entwicklungszeitraum optimistisch auf zwei Jahre, so könnte das System erst nach etwa vier Jahren in größerem Umfang industriell genutzt werden, weil der TÜV eine zweijährige Periode zum Nachweis der Betriebsbewährtheit vorschreibt. Vergleicht man schließlich noch die Kosten des Kaufs mit denen der Eigenentwicklung eines Echtzeitbetriebssystems, so scheidet diese Alternative auch aus offensichtlichen wirtschaftlichen Gründen sofort aus.

Aus obigen Ausführungen ergibt sich, daß der Einsatz eines portierbaren, ausgereiften und technisch anspruchsvollen Echtzeitbetriebssystems, das von einem unabhängigen Software-Haus entwickelt wurde und gepflegt wird, bei langfristiger Betrachtungsweise die wirtschaftlich optimale Lösung ist, sofern im eigenen Hause einige Spezialisten bereitstehen, die intime Kenntnisse des Systems besitzen, eigene Treiber entwickeln, die Anwendungsprogrammierer beraten und im Notfall eine größere Systemgruppe aufbauen können. Bei kurzfristiger Orientierung wird sich eigene Kompetenz bzgl. eines Echtzeitbetriebssystems kaum schnell und kostengünstig erwerben lassen. Deshalb empfiehlt es sich, unter großem Zeitdruck durchzuführende Projekte an ein Software-Haus zu vergeben, das auf diesem

Anwendungsgebiet erfahren und mit dem gewählten Echtzeitbetriebssystem vertraut ist. Auf jeden Fall sollte man immer auf die Eigenentwicklung eines Echtzeitbetriebssystems verzichten.

13.1.5 Wirtschaftlichkeit der Programmwartung

An jedes Betriebssystem sind sehr hohe Anforderungen bezüglich seiner Zuverlässigkeit zu stellen. Ein Betriebssystem muß sehr gründlich ausgetestet sein – dazu gehört auch ein längerer praktischer Einsatz – damit möglichst sichergestellt werden kann, daß die Fehlersuche eines Anwenders nicht auch auf die Betriebssystemroutinen ausgedehnt werden muß.

Ein Betriebssystem ist passiv, da es vom Anwender beauftragt wird und entsprechende Dienste zu leisten hat. Für die Wartung der Anwenderprogramme ist es wesentlich, daß ein Betriebssystem an seiner Aufrufschnittstelle besonders robust ist. Es muß schon im Entwurf große Sorgfalt darauf verwendet werden, daß nur auf Plausibilität geprüfte Aufträge (alle Parameter und Parameterkombinationen sind zu betrachten) in das System gelangen. Schwächen an dieser Stelle verursachen mit Sicherheit irgendwann einmal gravierende Fehlreaktionen im Betriebssystem, die deshalb besonders unangenehm sind, weil sie keinen Hinweis auf die Fehlerquelle mehr enthalten. Leider ist der Entwurf eines Betriebssystems sehr schwer zu prüfen, und leider treten Fehler der beschriebenen Art erst beim intensiven Umgang mit einem System auf.

Es versteht sich von selbst, daß jeder vom Betriebssystem entdeckte Fehler in angemessener Weise gemeldet werden muß, d.h. der Anwender muß hinreichende Informationen über die Fehlerursache erhalten. Fehlermeldungen sind auftrags- oder systembezogen. Letztere müssen auf einer Konsole ausgegeben werden, falls eine solche vorhanden ist. Falls nicht oder falls ein Fehler weitreichende Folgen haben kann, ist dringend zu verlangen, daß das Echtzeitbetriebssystem ein Fehlersignal an die Anwendung abgibt, so daß diese eine entsprechende Fehlerreaktionen einleiten kann. Das schon angesprochene BITE-Programm und seine Unterstützung durch das Echtzeitbetriebssystem liefern weitere günstige Voraussetzungen für die Programmwartung, indem sie das System in die Lage versetzen, auch auf Geräteprobleme zu reagieren. Bei der Fehlersuche gibt es wohl nichts Unangenehmeres als Gerätefehler.

Generell muß gefordert werden, daß einem Echtzeitbetriebssystem alle Mittel zur Verfügung gestellt werden, um Fehler zu erkennen. Dazu gehört auch die Ausschöpfung sämtlicher Sicherungsmaßnahmen, die Prozessoren anbieten. Erwähnt sei hier der Protected Mode, der es erlaubt, fehlerhafte Speicherzugriffe zu erkennen. Die Ablage von Code und konstanten Daten in Festwertspeichern bringt einen weiteren Sicherheitsgewinn. Ein Echtzeitbetriebssystem muß ROM-fähig sein.

Betrachtet man die durch aufgetretene Fehler oder durch geänderte Anforderungen bedingte Programmevolution und die damit verbundenen Risiken einerseits und die geforderte Zuverlässigkeit des Gesamtsystems andererseits, so gelangt man zur Forderung nach einem streng modularen Entwurf, der alle – auch interne – Schnittstellen offenlegt, klar definiert und der weiterhin durch Einfachheit besticht. Nur so kann sichergestellt werden, daß ein Gesamtsystem überschaubar bleibt, daß es somit zu einer Programmevolution fähig ist, daß die potentiellen Fehler bei einer Änderung keine großen Auswirkungen haben – oder auch, daß das System schlichtweg wartbar ist.

Wie bereits erwähnt, können langfristige Investitionen in ein Programmsystem nur dadurch geschützt werden, daß man sich gegenüber geänderter Hardware entsprechend flexibel ver-

hält. Die Geräteeigenschaften müssen parametrisierbar sein, wo immer dies möglich ist, und sie müssen in einer klar definierten, dünnen geräteabhängigen Programmschicht abgefangen werden, wo dies erforderlich ist. Sind diese Voraussetzungen nicht gegeben, so muß bei jeder Anpassung an geänderte Hardware mit umfangreichen und höchst bedenklichen Manipulationen an größeren Programmteilen gerechnet werden, was mit ziemlicher Sicherheit immer wieder auch erhebliche Wartungsprobleme nach sich zieht.

Die fortschreitende Geräteentwicklung (Leistungssteigerung der Prozessoren) und die ebenso weiterentwickelten hochoptimierenden Übersetzer höherer Programmiersprachen verbieten es heute auch für Echtzeitsysteme, auf niedere Programmiersprachen oder gar Assembler-Programmierung zurückzugreifen. Die Vorteile höherer Programmiersprachen bei der Entwicklung und Wartung können und müssen ausgenutzt werden. Für ein Echtzeitbetriebssystem ist daher zu fordern, daß es Laufzeitsysteme für gängige Programmiersprachen zur Verfügung stellt, die insbesondere die Echtzeitprogrammierung unterstützen. Beispiele solcher Sprachen sind etwa PEARL und Ada. Für den Umgang mit höheren Programmiersprachen sind leistungsfähige symbolische Testhilfsmittel zu fordern. Ohne solche Werkzeuge ist die Programmwartung darauf angewiesen, fällige Inspektionen des ablaufenden Codes anhand von Maschinenbefehlen zu vollziehen. Ausgehend von der vorliegenden höheren Programmiersprache sind zu Maschinenbefehlen aber nur schwer Bezüge herzustellen. Ein für ein Echtzeitbetriebssystem verwendbares Testwerkzeug muß darüber hinaus noch weitere Forderungen erfüllen. Es muß die Betriebssystemobjekte, mit denen der Anwender umgeht, identifizieren und deren Zustand darstellen können. Selbstverständlich muß verlangt werden, daß die Betriebssystemobjekte symbolisch ansprechbar sind.

13.1.6 Struktur eines portierbaren Echtzeitbetriebssystems

Aufbauend auf den in diesem Abschnitt zusammengestellten anwenderorientierten Forderungen soll nun kurz die Struktur eines Echtzeitbetriebssystems für Einprozessorsysteme vorgestellt werden, das auch für eingebettete Anwendungen geeignet ist. Diese Struktur gewährleistet leichte Konfigurierbarkeit, um eine große Vielfalt von Hardware-Architekturen zu unterstützen, die von Industrie-PCs über Einplatinenrechner bis hin zu hochspezialisierten Systemen reicht.

Neben den Basisdiensten Prozeß-, Ereignis- und Zeitverwaltung bietet das System Mechanismen zur Ein- und Ausgabe, zur Dateiverwaltung und zur Mensch-Maschine-Kommunikation an. Es ist konfigurierbar und kann von einem sehr kleinen Kern mit nur wenigen Funktionen aus zu einem Betriebssystem mittleren oder großen Funktionsumfangs erweitert werden. Eine Umgebung zur Programmierung von Anwendungsprogrammen in Form von Dienstprogrammen und Übersetzern wird nicht geboten. Zu diesem Zweck sind eigene Entwicklungsrechner einzusetzen.

Wie in Bild 13.1 dargestellt, hat das System eine Schichtenarchitektur, die sich grundlegend von der in Abschnitt 4.3 beschriebenen unterscheidet. Sie besteht aus einem geräteabhängigen Kern und einer geräteunabhängigen Schale. Der Kern ist für die nebenläufige Prozeßausführung, Kommunikation und Synchronisation von Tasks sowie für die Zeit- und Ereignisverwaltung, dynamische Zuweisung und Freigabe von Hauptspeichersegmenten und primitive Ein- und Ausgabe verantwortlich. Die Schale ist als eine Menge von Prozeduren organisiert, die von Anwendungsprozessen aufgerufen werden können und dann als Teile dieser Tasks ausgeführt werden. Die von der Schale angebotenen Funktionen umfassen geräteunabhängige Ein- und Ausgabe hohen Niveaus, Dateiverwaltung und Ereigniseinplanung. Ein Sprachprozessor bildet eine zusätzliche Schicht. In ihr sind sprachspezifische

Operationen implementiert, wodurch die Basis geschaffen wird, um in höheren Programmiersprachen geschriebenen Code auszuführen.

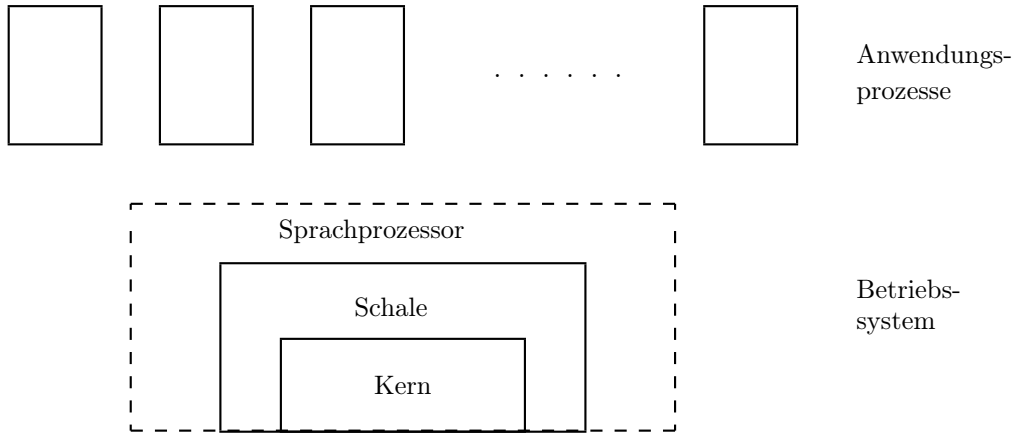

Bild 13.1: Architektur eines portierbaren Echtzeitbetriebssystems

Die Struktur von Anwendungsprozessen ist insoweit statisch, als Tasks nur in einer Initialisierungsphase erzeugt werden können, ehe die Anwendungsprogramme gestartet werden. Dies spart sowohl Speicherplatz für Code als auch Programmausführungszeit. Jeder Task wird eine Antwortzeit zugewiesen, welche während ihrer Ausführung dynamisch geändert werden kann. Das Task-Zustandsübergangsdiagramm ist das gleiche wie das in Abschnitt 4.3 als Bild 4.8 eingeführte.

Eine zusätzliche Komponente des Systems ist eine Testhilfe, die die Ausführung von System- und Anwendungsprozessen zu verfolgen und zu überwachen erlaubt. Entsprechend der allgemeinen Systemarchitektur ist die Testhilfe nicht als Systemprozeß implementiert, sondern als Paket von Unterprogrammen, die nach Bedarf in die Rümpfe der Anwendungsprozesse eingebaut werden können. Während des Bedienerdialogs mit der Testhilfe ist der Zustand des gesamten Systems „eingefroren“, und es ist dem Anwender möglich, einen augenblicklichen „Schnappschuß“ des ganzen Systems einzufangen.

Der Systemkern implementiert drei Mechanismen zur Synchronisation von und zur Kommunikation zwischen Tasks: Nachrichtenpuffer, binäre Semaphoren und Ereignisse. Diese Objekte werden während der Systeminitialisierung statisch erzeugt. Jeder Task wird ein Nachrichtenpuffer zugeordnet, der nur vom Eigentümer gelesen werden kann, in den jedoch jede andere Task schreiben darf. Die Semantik der Nachrichtenweitergabeoperationen *Senden* und *Empfangen* wurde bereits in Abschnitt 4.4 definiert.

Schließlich wird die antwortzeit- oder termingesteuerte Zuteilung ablaufbereiter Tasks unterstützt, mit der wir uns in den folgenden Abschnitten dieser Kurseinheit eingehender beschäftigen wollen.

Selbsttestaufgabe 13.1-1:

Informieren Sie sich unter `http://www.rtos-uh.de` über das betriebsbewährte Echtzeitbetriebssystem RTOS-UH und stellen Sie fest, inwieweit es den in diesem Abschnitt aufgestellten Anforderungen genügt.