Echtzeit-Betriebssysteme für Multimikrorechnersysteme (c) 1985

Prinzipien

Um die parallele Ausführung verschiedener Anwendungsprogramme und das Zusammenwirken der Funktionseinheiten eines Multimikrorechnersystems zu gewährleisten, sind Softwarevorkehrungen erforderlich, die in ihrer Gesamtheit als Echtzeitbetriebssysteme bezeichnet werden. Zumeist ist das Multimikrorechnersystem aus der Sicht des Anwenders bzw: Kunden kein frei programmierbarer Universalrechner, sondern ein „schlüsselfertiger" Hardware-Software-Komplex (Turnkey System), der die gewünschten Gebrauchseigenschaften realisiert (z. B. als peripheres EDV-Gerät, als Maschinensteuerung usw.). Somit handelt es sich darum, daß eine umfangreiche Anwendungssoftware auf derartigen Systemen zwar abgearbeitet, aber in der Regel nicht entwickelt wird. Die betreffenden Echtzeitbetriebssysteme, die im folgenden näher betrachtet werden, umfassen deshalb Funktionen zum Starten von Anwendungsprogrammen, zum Zugriff auf Datenbereiche, zur Synchronisation von Abläufen usw., aber keine Mittel zur Entwicklung von Programmen (Assembler, Compiler usw.).

Zu den wesentlichen Aspekten gehören die Laufzeiteffizienz, die Funktionssicherheit und die Beherrschbarkeit im Entwicklungsprozeß. Da Multimikrorechnersysteme oft deshalb benutzt werden, weil der einzelne Mikrorechner nicht leistungsfähig genug ist und weil ökonomische Vorteile (Hardwareaufwendungen, Produktionskosten) gegenüber einer reinen Speziallösung realisiert werden sollen, besteht die Notwendigkeit, zumindest bei den leistungsentscheidenden Softwarekomponenten Wert auf eine hohe Laufzeiteffizienz zu legen. Zum einen zwingt dies zum maschinennahen Programmieren (Assemblerniveau) dieser leistungsentscheidenden Komponenten. Zum anderen sind Betriebssysteme, die ohne umfangreiche Anpassungsarbeiten sofort lauffähig sind, kaum verfügbar, so daß man auch in dieser Hinsicht auf eigene Entwicklungsleistungen angewiesen ist.

Die Aufteilung der Software in Anwendungsprogramme und Betriebssystem verlängert zunächst den Entwicklungsprozeß, da erst mit dem Programmieren der Anwendungsabläufe begonnen werden kann, wenn das Betriebssystem spezifiziert worden ist. Andererseits sind selbst einfache Aufgaben kaum beherrschbar, wenn die Trennung nicht vollzogen wird. Man hat die Wahl, den als Betriebssystem abgetrennten Softwarekomplex mit flexiblen und bequem zu nutzenden Funktionen auszustatten oder ihn eher elementar auszulegen. Je leistungsfähiger (im funktionellen Sinne) das Betriebssystem ist, um so schwieriger wird dessen Erprobung, um so einfacher die Anwendungsprogrammierung.

Die Laufzeiteffizienz eines leistungsfähigeren (d. h. komplexeren) Betriebssystems ist oft schlechter als die eines eher elementaren. Der Leistungsverlust kann aber minimal gehalten werden, wenn die kritischen Abläufe (jene, die man auch für ein elementares Betriebssystem als unverzichtbar ansehen würde) entsprechend sorgfältig programmiert werden und wenn für bestimmte Sonderfunktionen (z. B. Plausibilitäts- bzw. Verträglichkeitsprüfungen von Parametern) Abschaltmöglichkeiten oder dergleichen vorgesehen werden. Des weiteren tritt insgesamt (aus Anwendersicht) kaum Leistungsverlust auf, wenn das Betriebssystem Funktionen realisiert, die sonst in den Anwendungsprogrammen direkt enthalten sein müßten. (Die Quelle von Leistungsminderung ist in der Regel lediglich die standardisierte Parameterübergabe und der Systemaufruf.)

Andererseits erfordert ein reichhaltig ausgestattetes Betriebssystem eine umfangreiche Erprobung, denn alle vorgesehenen Funktionen müssen getestet werden. Die Vielzahl der möglichen Kombinationen des Aufrufs, der Parameterübergabe usw. ist kaum erschöpfend zu testen. Damit muß bei der Anwendungserprobung mit Seiteneffekten, Fehlfunktionen u. ä. gerechnet werden. Diese machen sich oft als sporadische Funktionsstörungen bemerkbar. Das Entwickeln der Software auf anderen Einrichtungen (Entwicklungssystem, Cross-Assembler usw.) führt zu weiteren Problemen. Die Betriebssystemkomponenten realisieren viele Funktionen, die auch in modernen höheren Programmiersprachen vorgesehen sind. (Umgekehrt gibt das Studium neuerer Entwicklungen der höheren Programmiersprachen viele Anregungen für Auswahl und Gestaltung der Funktionen eines Betriebssystems.) Im Gegensatz zu höheren Programmiersprachen existiert jedoch kein Compiler, und oft sind auch die verfügbaren Entwicklungshilfen den Besonderheiten von Multimikrorechnersystemen nicht angemessen, so daß komplexe Sprachkonstruktionen mit Mitteln der Assemblersprache formuliert werden müssen. Dies allein bereitet erfahrungsgemäß kaum Schwierigkeiten. Schwerwiegender hingegen ist, daß Aktivitäten, die bei ausgebauter Entwicklungsunterstützung etwa ein Compiler leisten kann, bis zur Laufzeit verschoben werden müssen. Eine gewisse Entlastung kann dadurch erreicht werden, daß in der Konzeption zwischen Laufzeit und Etablierungszeit unterschieden wird. Es muß von vornherein festgelegt werden, welche Programme, Datenbereiche, Steuerinformationsblöcke usw. statisch placiert werden können (fest in ROMs oder durch initiale Lade- und Etablierungsabläufe) und welche dynamisch verwaltet werden müssen, da die dynamische Verwaltung mehr Aufwand zur Laufzeit kostet.

Klare und übersichtliche Schnittstellen zwischen Anwendungsprogrammen und Betriebssystem sind für folgende Aspekte von besonderer Bedeutung:

- Auslösen, Starten, Synchronisieren und Beenden von Programmabläufen
- Zugriff auf Programme und Datenbereiche
- zeitmultiplexer Ablauf mehrerer Programme.

Es ist empfehlenswert, diese Probleme zunächst auf einem relativ abstrakten Niveau zu betrachten und zu versuchen, einen hohen Abstraktionsgrad bis zur Implementierung durchzuhalten. Im Verlauf der schrittweisen Verfeinerung der Konzeption (bis zur Kodierung) wird der Grad der Abstraktion ebenfalls schrittweise (abhängig von den konkreten Erkenntnissen und Anforderungen) reduziert. Wesentliche Gründe, eine abstrakte Betrachtungsweise aufzugeben (zugunsten einer u. U. sehr speziellen Implementierung eines Konzepts), können Laufzeiteffizienz, Speicherbedarf und Anwendungsbedarf (wenn z. B. nur eine spezielle Teilfunktion benötigt wird) sein.

So werden in einem realisierten Betriebssystem folgende grundlegende Abstraktionen benutzt:

- Objekte für den Zugriff zu Informationseinheiten wie Programmen und Datenbereichen (objektorientierte Betrachtungsweise)
- Partitions für die Organisation des Multiprogramming
- Ereignisse für das Auslösen bzw. Synchronisieren von Programmabläufen.

Wesentliche Gesichtspunkte dazu sollen in den folgenden Abschnitten beschrieben werden.

Objektstruktur

Jeder zusammenhängende Informationsblock (Programm, Datenbereich usw.) wird als Objekt bezeichnet. Der Zugriff zu solchen erfolgt üblicherweise durch direkte Angabe von Adressen. Eine wirkungsvolle Abstraktion besteht darin, statt der Adressen Objektidentifizierer anzugeben. Ein solcher Identifizierer ist die Ordinalzahl des betreffenden Objekts aus der Menge aller Objekte. Die einfachste Weise, ihn festzulegen, ist das Durchnumerieren aller Objekte. Da für konkrete Zugriffe natürlich Adressen gebraucht werden, muß man über ihn die jeweilige Adresse aus einer Tabelle ermitteln (Bild 1).

Dieses Verfahren ist in Laufzeitsystemen höherer Programmiersprachen gebräuchlich. In Rechenanlagen wurden schon vor längerer Zeit gelegentlich Hardwaremittel zur Unterstützung dieses Prinzips vorgesehen. Neuerdings sind auch Mikroprozessoren im Sinne dieser objektorientierten Architektur realisiert worden. Muß bei jedem Zugriff eine Tabelle benutzt werden, so sinkt die Laufzeiteffizienz selbst bei Hardwareunterstützung oft intolerabel ab. Bei üblichen Mikroprozessoren kann die Leistungsminderung nicht akzeptiert werden.

Hingegen hat sich eine andere Lösung bewährt:

- Die Objektstruktur wird im wesentlichen dazu benutzt, Programme unabhängig voneinander entwickeln zu können und

den Zugriff zu Datenbereichen zu ermöglichen. Neben residenten Objekten sollen auch transiente Objekte vorgesehen werden, also solche, die von einem externen Datenträger oder einem anderen Speicherbereich erst dann in den zur Ausführung vorgesehenen Bereich geladen werden, wenn dies erforderlich ist. Da übliche Mikroprozessoren (z. B. U 880) keine Vorkehrungen für Adressenverschiebung zur Laufzeit haben, muß der verfügbare Speicherplatz stets statisch aufgeteilt werden. Somit liegt für jedes Objekt fest, in welchem Bereich es zur Laufzeit (d. h., wenn es benötigt wird) verfügbar sein muß.

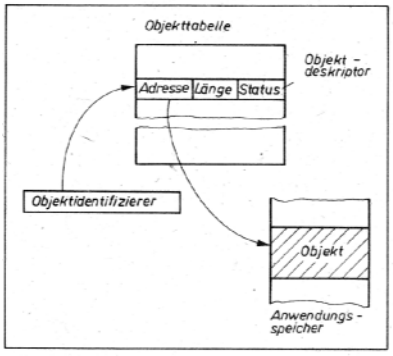

Bild 1: Prinzip der Ermittlung der Objektadresse aus dem Objektidentifizierer

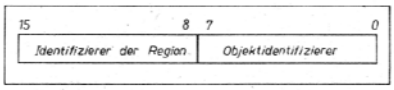

Bild 2: Interpretation eines 16-bit-Objektidentifizierers

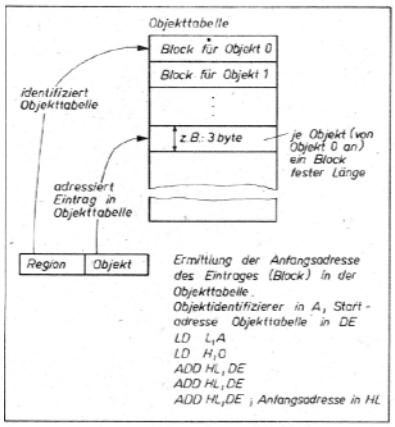

Bild 3: Adressierung der Objekttabelle durch den Objektidentifizierer

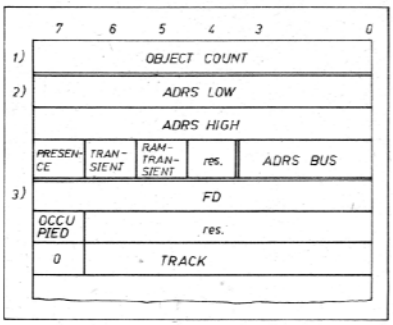

Bild 4: Detaillierter Aufbau einer Objekttabelle. 1) Gibt Anzahl der Objekte an, für welche die Objekttabelle eingerichtet ist (Gesamtlänge der Objekttabelle = OBJECT COUNT · 3 + 1); 2) Eintrag für Objekt 0; 3) Eintrag für Objekt 1 als Beispiel eines nicht residenten Objekts

- Die Anzahl der Objekte wird nicht allzu groß sein, und jeder Mikrorechner wird nur auf einen Teil der Objektmenge zugreifen. Somit ist ein 16-bit-Identifizierer bei weitem ausreichend (max. 64 K Objekte). Dieser Objektraum wird in 256 Regionen zu 256 Objekte unterteilt. Damit ergibt sich eine einfache Interpretation des Identifizierers gemäß Bild 2. Für jede Region gibt es eine Objekttabelle (Object Reference Table ORT), die durch das niedere Byte des Objektidentifizierers einfach adressiert werden kann (Bild 3).
  Oft ist es ausreichend, für jeden Mikrorechner eine ORT vorzusehen (lokale Region). Sind 256 Objekte für einen Mikrorechner zu wenig, so reicht oft eine Aufteilung des lokalen Objektraumes in zwei Regionen aus: Für Programm- und Datenbereichs-Objekte sind jeweils separate Objekttabellen vorgesehen (PRT, DART).
  Sind Zugriffe zu Objekten erforderlich, die nicht in den lokalen Tabellen zu finden sind, so muß an zentraler Stelle eine Tabelle der Objekttabellen angeordnet werden.
- Wird ein Objekt zur Laufzeit erstmalig benötigt, so ist eine Systemfunktion (z. B. GET object-ident) auszuführen, um den Zugriff zu gewährleisten. Anschließend wird der betreffende Speicherbereich auf übliche Weise adressiert. Ein konkurrierendes Benutzen des gleichen Speicherbereichs durch mehrere im Multiprogramming-Betrieb laufende Programme ist nicht möglich (d. h., dies muß durch die Programmierdisziplin gewährleistet werden; eine Kontrolle zur Laufzeit hätte einen untragbaren Leistungsverlust zur Folge).
- Im Detail ist eine Objekttabelle nach Bild 4 aufgebaut. Die Möglichkeiten für die Speicherung der Objekte sind in der Tafel 1 angegeben.
  Die Objekttabelle gibt stets die aktuelle Position des Objekts an. Ist dies nicht präsent (d. h. nicht in dem Teil des Speichers, wo es zur Laufzeit benötigt wird), so enthält die Tabelle die direkte Position (im RAM bzw. auf externen Datenträgern), wo das Objekt zu finden ist. Auf dieser Position ist die weitere beschreibende Information (Speicheradresse, Länge, Status usw.) verzeichnet. Besonders im Falle externer Datenträger wird dadurch, daß die Position des Objekts direkt angegeben ist (und nicht etwa ein Dateiname, mit dem ein Verzeichnis durchsucht werden muß), die Zugriffszeit wesentlich verringert (s. Bild 5).
- Für laufzeitkritische Abläufe müssen alle betreffenden Objekte ständig präsent sein. In diesem Fall wird das beschriebene Zugriffsverfahren nur zur Initialisierung benutzt (z. B. beim Anfangsladen, um die Objekte in den Speicher zu transportieren).

Multiprogramming-Organisation

Um in einem Mikrorechner n Programme zeitmultiplex abarbeiten zu können, sind n Partitions vorzusehen. Jede Partition ist durch einen Steuerbereich (Partition Control Area, PCA) und einen Stackbereich gekennzeichnet.
In einer U-880-Implementierung umfaßt die PCA mindestens 128 byte und höchstens 255 byte. Dabei sind 128 byte für Steuerzwecke reserviert; der verbleibende Bereich kann anwendungsspezifisch belegt werden (Steuerregister, Adressenregister, Zustandsbits, Selektoren usw.). Das IY-Register zeigt fest auf die PCA; es darf nicht von Anwendungsprogrammen überladen werden. Mit dieser Konvention können reentrante Programme vorgesehen werden, d. h., ein Programm ist in einem Mikrorechner in mehreren Partitions lauffähig, wenn alle Bezugnahmen zu Arbeitsbereichen u. ä. in der Form (IY + displacement) angegeben werden.
Bild 6 zeigt die prinzipielle Aufteilung einer PCA.
Des weiteren sind die Austauschregister des U 880 für die schnelle Interruptbehandlung reserviert und somit für normale Anwendungsprogramme nicht verfügbar.
Tafel 2 gibt einen Überblick über wesentliche Zustände, die eine Partition haben kann.

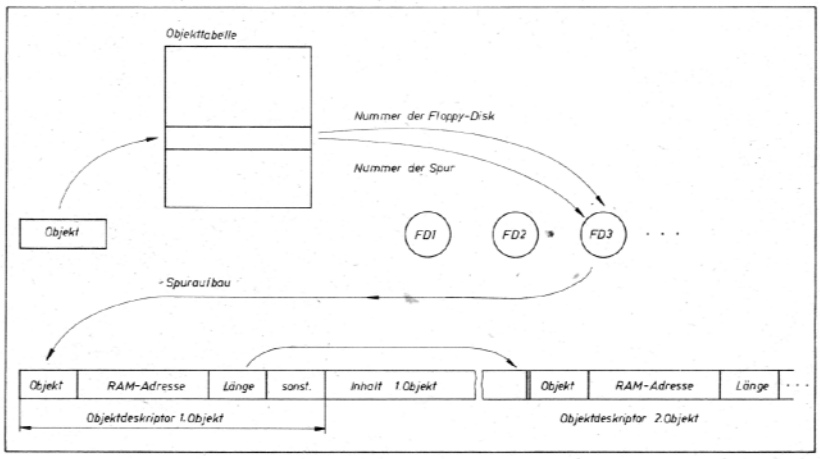

Bild 5: Prinzip des Zugriffs zu Objekten, die sich auf externen Datenträgern befinden.
Hinweis: Der gesamte Inhalt der Spur wird in einen Spurpuffer im RAM transportiert. Dort wird der Objektdeskriptor des betreffenden Objektes gesucht

Tafel 1: Möglichkeiten zur Speicherung von Objekten

Art des Objekts	Speicherung	Eintrag in Objekttabelle	Bemerkungen
nicht vorhanden	—	alle drei Bytes = 0	—
transient, nicht im RAM (nicht resident)	auf Floppy-Disk	Identifikation der Floppy-Disk und der Spur; OCCUPIED = 1 (zeigt an, daß der 3-byte-Block belegt ist); PRESENCE = 0, der Rest dieses Bytes enthält die Spurangabe	Auf angegebener Spur der Floppy-Disk befinden sich Deskriptorblöcke, die RAM-Adresse, Länge usw. des Objekts enthalten
transient, im RAM (resident)	in einem Transientbereich des RAM; jedem Transientbereich ist ein Steuerblock von 19 byte Länge vorgeordnet, der die Verwaltungs- und Belegungsinformation enthält	Anfangsadresse des Transientbereiches, PRESENCE und TRANSIENT sind gesetzt	Zugriffssteuerinformation für die Floppy-Disk befindet sich im Steuerblock des Transientbereichs
RAM-transient	in einem beliebigen Bereich des RAM; die ersten beiden Bytes des dort gespeicherten Objekts geben dessen Länge an	Anfangsadresse des RAM-Bereiches, aus dem das Objekt geholt wird; TRANSIENT und RAM TRANSIENT sind gesetzt; befindet sich das Objekt im Transientbereich, so ist PRESENCE gesetzt, und die Adresse ist die Anfangsadresse des Transientbereichs	Zeiger auf den Transientbereich befindet sich auf einer festen Position der aktuellen PCA
resident	ständig im RAM	Anfangsadresse im RAM, PRESENCE ist gesetzt	—

Tafel 2: Wesentliche Zustände einer Partition

Zustand	Kennzeichen, Bemerkungen	erreicht durch	verlassen durch
Ruhe		Anfangsrücksetzen, TERMINATE-Anweisung	Ereignis-Auslösung (I- oder C-Ereignis)
aktiv mit Laufzeit	Partition hat Laufzeit, IY zeigt auf PCA, SP zeigt auf den Stackbereich	Ereignisauslösung, Zeitzuteilung durch TIME SLICING, Aufheben der Suspendierung (WAKE UP), Ablauf eines DELAY-Intervalls	TERMINATE-Anweisung, Zeitzuteilung durch TIME SLICING, Ereignisauslösung für andere Partition, verschiedene Systemanweisungen
aktiv ohne Laufzeit	Partition hat keine Laufzeit, ist aber aktiv	Zeitzuteilung durch TIME SLICING, Ereignisauslösung für andere Partition	Zeitzuteilung (TIME SLICING oder Deaktivieren anderer Partitions)
suspendiert	Für die Dauer der Suspendierung erhält die Partition nie Laufzeit; Ereignisse werden jedoch angenommen	Systemanweisung SUSPEND PARTITIONS	Systemanweisung WAKE UP PARTITIONS, Ereignis vom Typ C
verzögert	Die Partition wird für die Dauer eines vorgegebenen Zeitintervalls Laufzeit entzogen	Systemanweisung DELAY	Ablauf des Zeitintervalls, Ereignis vom Typ C

Eine Partition kann prinzipiell nur durch Ereignisauslösung aktiviert werden. Eine Rückkehr zum Ruhezustand ist nur durch entsprechende Systemfunktionen (TERMINATE, CANCEL) möglich. Bei Aktivierung erhält die betreffende Partition sofort Laufzeit. Die Laufzeit wird durch Aktivierung einer anderen Partition (infolge Auslösung eines anderen Ereignisses), SUSPEND-Anweisungen, WAIT-Anweisungen, DELAY-Anweisungen und TIME SLICING wieder entzogen.

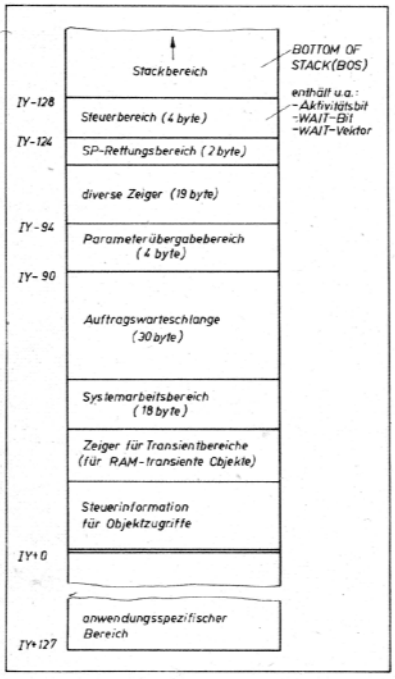

Bild 6: Aufteilung einer PCA

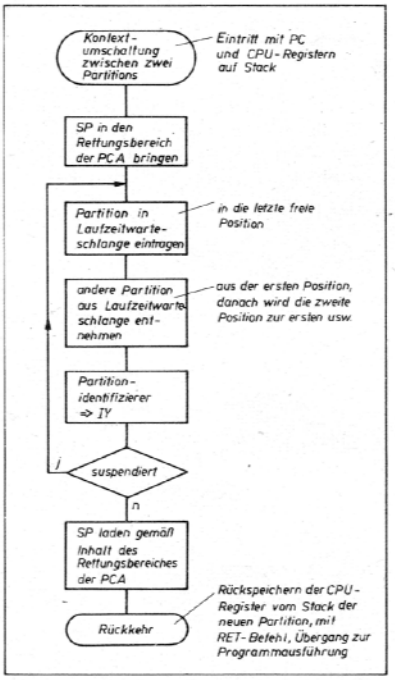

Bild 7: Ablauf der Kontextumschaltung zwischen zwei Partitions

Üblicherweise (mit Ausnahme von WAIT und DELAY) wird eine Partition, der Laufzeit entzogen wurde, am Ende der Laufzeitwarteschlange eingeordnet. Wird eine Partition inaktiv bzw. wird TIME SLICING wirksam, so erhält die Partition in der ersten Position der Laufzeitwarteschlange Laufzeit. Diese „Kontextumschaltung" ist im Bild 7 dargestellt.

Ist in einem Mikrorechner TIME SLICING eingeschaltet, so wird zyklisch alle 30 ms der aktuellen Partition die Laufzeit entzogen, diese am Ende der Laufzeitwarteschlange angefügt und der ersten Partition aus der Laufzeitwarteschlange Laufzeit zugeteilt, so daß alle aktiven Partitions zyklisch Laufzeit in Scheiben von 30 ms Dauer erhalten. Dieses einfache Prinzip des zyklischen Weiterschaltens ist für die meisten Verarbeitungsprobleme angemessen. Kompliziertere Strategien der Zeitzuteilung (z. B. mit verschiedenen Prioritäten der einzelnen Partitions) sind nicht notwendigerweise effektiver. Die Probleme der gegenseitigen Behinderung (engl. Deadlock) werden schwieriger beherrschbar (bei Round Robin tritt theoretisch kein Deadlock auf, wenn die einzelnen Partitions unabhängig voneinander sind). Weiterhin ist zu bedenken, daß jede Zuteilungstrategie ihrerseits programmiert werden muß. Dafür stehen auch nur die üblichen Befehle zur Verfügung. Somit wächst der Laufzeitbedarf des Betriebssystems mit zunehmender Kompliziertheit der Zuteilungsstrategie. Dadurch wird der beabsichtigte Gewinn an Effizienz oft wieder beeinträchtigt.

Ereignissteuerung

Der Begriff „Ereignis" kennzeichnet allgemein das Auftreten einer Bedingung. Es gibt physische und logische Ereignisse. Physische Ereignisse sind Bedingungen, die in peripheren Einrichtungen auftreten (z. B. Betätigen von Tasten, Anforderungen von einem zu steuernden Interface, Fehlersignale peripherer Geräte usw.). Das Betriebssystem verarbeitet nur logische Ereignisse. Ein logisches Ereignis ist durch einen Ereignissteuerblock (Event Control Block, ECB) charakterisiert. Die physischen Ereignisse sind somit in logische umzusetzen. Dazu sind spezifische Behandlungsroutinen vorzusehen. Ist eine periphere Einrichtung an die E-A-Schaltungen eines Mikrorechners angeschlossen, so führen physische Ereignisse in der Regel zu Interrupts. Für jede Interruptursache muß eine Behandlungsroutine vorgesehen sein, die den betreffenden ECB adressiert und zur logischen Ereignisbehandlung des Betriebssystems verzweigt. Werden physische Ereignisse von autonomen Schaltmitteln registriert, so können diese Einrichtungen Interrupts über den Systembus auslösen, wobei der übertragene Interruptvektor den zugehörigen ECB identifiziert (Bild 8). Die Struktur des ECB ist im Bild 9 dargestellt.

Von besonderer Bedeutung sind die Angaben über den Typ des Ereignisses, über die betroffene Partition und über das zu aktivierende Programm. In bestimmten Ereignissen können weiterhin im Rahmen der Auslösung Parameter übergeben werden (maximal vier Bytes). Die Anzahl ist für viele Anwendungsfälle ausreichend. Sind vier Bytes für die aktuelle Parameterinformation zu wenig, so reicht diese Anzahl auf jeden Fall, um einen Adressenzeiger auf den eigentlichen Parameterblock sowie zusätzliche Funktionssteuerinformation zu übertragen. Das höchstwertige Bit des ersten Parameterbytes ist das Free-Bit. Dies wird vor der Parameterübertragung und Ereignisauslösung im Rahmen von Test-and-Set-Abläufen dazu benutzt, festzustellen, ob der Parameterbereich frei ist und das Ereignis ausgelöst werden kann. Das jeweilige Anwendungsprogramm findet die übergebenen aktuellen Parameter auf festen Positionen in der aktuellenen PCA vor (vgl. Bild 5).

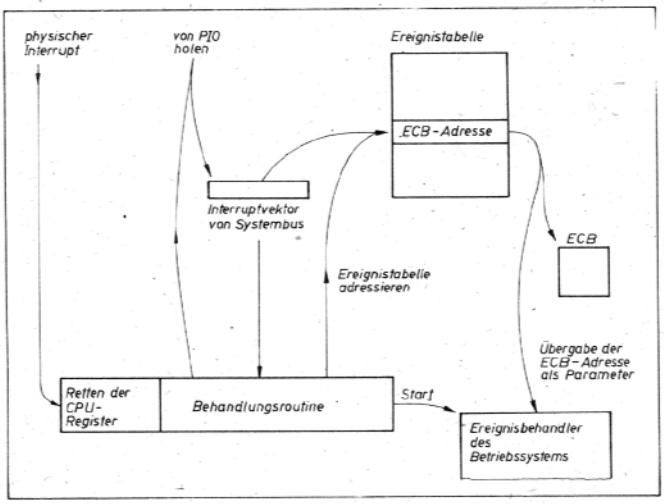

Bild 8: Ereignisauslösung durch physischen Interrupt

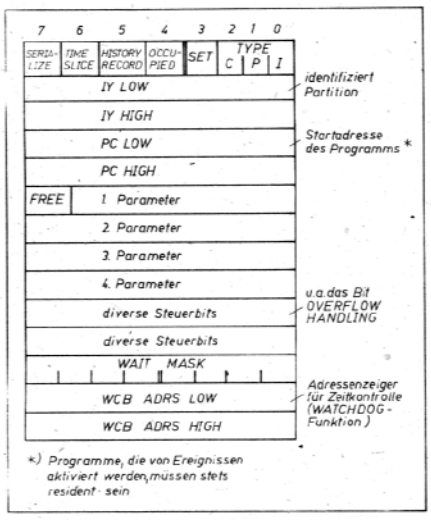

Bild 9: Struktur eines Event Control Block (ECB)

Es gibt drei Typen logischer Ereignisse:

- Typ I (Initiate)
  Ein solches Ereignis veranlaßt den Start eines Programms (gemäß dem angegebenen Befehlszähler) in der angegebenen Partition (diese wird aktiviert). Ist die Partition bereits aktiv, so wird die Programmstartinformation (Befehlszähler + aktuelle Parameter) in eine Auftragswarteschlange eingetragen, die in der PCA geführt wird. Ist diese Warteschlange, die vier Positionen enthält, bereits voll, so gibt es in Anhängigkeit vom Steuerbit OH (Overflow Handling) im ECB zwei alternative Reaktionen.
  1. Ist OH = 0, so wird das physische Auslösen weiterer Ereignisse verhindert, indem der RETI-Befehl für das Beenden der Interruptbehandlung zunächst nicht ausgeführt wird, und das aktuelle Ereignis wird in eine spezielle Ereigniswarteschlange eingetragen, die im Steuerbereich des Mikrorechners gehalten wird. Diese wird vom Betriebssystem zyklisch inspiziert, so daß beim Freiwerden der Auftragswarteschlange das Ereignis behandelt werden kann. Das Entnehmen des Ereignisses aus der Ereigniswarteschlange ist mit der Abarbeitung eines RETI-Befehls verbunden, so daß das Auslösen weiterer Ereignisse wieder möglich wird. Ist hingegen OH = 1, so wird die Ereignisauslösung ignoriert, und es wird eine Überlaufwarnung angezeigt (dies ist z. B. eine typische Reaktion bei Ereignissen, die auf manuelle Auslösung zurückgeführt werden können, etwa auf Tastenbetätigungen).
- Typ C (Cancel)
  Es wird ein Programm mit dem angegebenen Befehlszähler in der angegebenen Partition gestartet. Der Programmstart erfolgt ohne Rücksicht auf den Zustand der Partition. Diese wird aus Warteschlangen und anderen Steuerblöcken entfernt. Die Auftragswarteschlange wird gelöscht.
- Typ P (Proceed)
  Mit diesem Ereignistyp können Abläufe synchronisiert werden. Die Wirkung hängt mit einer WAIT-Anweisung des Betriebssystems zusammen. Wird ein solches Ereignis ausgelöst und die Partition befindet sich nicht im WAIT-Zustand, so wird lediglich die Tatsache der Auslösung im ECB vermerkt (SET-Bit wird eingeschaltet). Eine WAIT-Anweisung lädt einen WAIT-Vektor in die aktuelle PCA und fragt die angegebenen Ereignisse daraufhin ab, ob sie bereits aufgetreten sind. Bei jedem aufgetretenem Ereignis (SET = 1) wird der WAIT-Vektor mit der WAIT-Maske des ECB konjunktiv verknüpft, und das Resultat wird als neuer WAIT-Vektor gespeichert. Ist dieser gleich Null, wird die Programmabarbeitung fortgesetzt. Ansonsten gelangt die Partition in den WAIT-Zustand.
  Tritt das Ereignis auf, wenn sich die Partition in diesem Zustand befindet, so erfolgt ebenfalls die beschriebene Verknüpfung von WAIT-Vektor und WAIT-Maske, worauf die Partition entweder im WAIT-Zustand verbleibt oder die Programmabarbeitung fortgesetzt wird (Partition wird wieder aktiv). Tafel 3 zeigt einige Varianten von Abläufen, die sich mit den beschriebenen Mitteln erreichen lassen.

Neben dem Umsetzen physischer Ereignisse in logische gibt es die Möglichkeit, durch Systemanweisungen logische Ereigeignisse direkt auszulösen (CAUSE für Ereignisse im selben Mikrorechner, INTERRUPT für Ereignisse in anderen Einrichtungen).

Die physische Interruptbehandlung kann im Sinne der Verkürzung von Reaktionszeiten

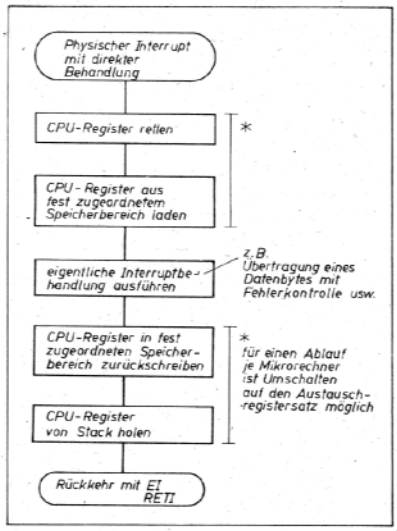

Bild 10: Ablauf einer Interruptbehandlung mit unmittelbarer Rückkehr

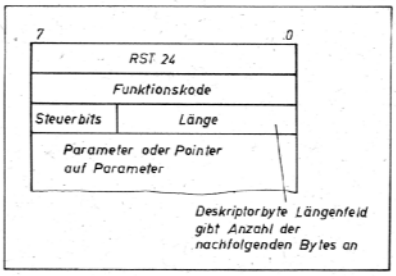

Bild 11: Parameterübergabe bei Systemruf über RST 24

auch die Ereignisauslösung umgehen. Dies ist dann möglich, wenn durch den Interrupt eine Behandlungsroutine aktiviert wird, die mit einer unmittelbaren Rückkehr beendet wird und anderen Abläufen keine Ressourcen (Speicherplatz, übermäßige Laufzeit) entzieht. Der Ablauf ist im Bild 10 dargestellt.

Damit läßt sich z. B. die Datenübertragung zu einem Druckwerk parallel zur Ausführung anderer Aufgaben organisieren. Allgemein werden physische Ereignisse in logische umgesetzt und als solche verarbeitet, wenn es darum geht, die Bearbeitung einer bestimmten Anwendungsaufgabe zu veranlassen. Innerhalb der Anwendungsaufgabe wird dann die jeweils effektivste Form der Kommunikation zwischen Programm und E-A-Einrichtungen verwendet.

Tafel 3: Beispiele für Abläufe, die durch Zusammenwirken von WAIT mit Ereignissen von Typ P möglich sind

	Beispiel 1	Beispiel 2	Beispiel 3
WAIT-Maske der Partition	0000 0001	0000 0011	0000 0011
WAIT-Vektor Ereignis 1	0000 0000	0000 0001	0000 0001
WAIT-Vektor Ereignis 2	0000 0000	0000 0010	0000 0010
WAIT-Vektor Ereignis 3	—	—	0000 0000
Beenden des WAIT-Zustandes	$E1 \vee E2$ (eines der beiden Ereignisse eingetroffen)	$E1 \wedge E2$ (beide Ereignisse eingetroffen)	$(E1 \wedge E2) \vee E3$ (Ereignisse 1, 2 zusammen eingetroffen oder Ereignis 3 eingetroffen)

Aufruf des Betriebssystems

Das Betriebssystem kann durch physische E-A-Behandlungsroutinen sowie durch explizite Aufrufe in Anwendungsprogrammen (Systemanweisungen) aktiviert werden. Für den ersten Fall ist eine Verzweigung zu einer Festadresse (für die logische Ereignisbehandlung) mit Parameterübergabe in den CPU-Registern vorgesehen. Für Systemanweisungen ist charakteristisch, daß es eine größere Anzahl davon gibt und daß verschiedene Varianten der Parameterübergabe wünschenswert sind. Weiterhin ist eine strikte Trennung zwischen Anwenderprogrammen und Betriebssystem sinnvoll (Entwickeln, Erproben, Ändern usw. sind dadurch unabhängig möglich). Deshalb ist der Aufruf über Funktionskode realisiert. Jede Systemanweisung wird durch einen Funktionskode identifiziert. Als Aufrufbefehle sind RST 24 und RST 32 vorgesehen. Bei RST 24 folgen der Funktionskode und (erforderlichenfalls) Parameter (bzw. Zeiger auf Parameter) dem Aufrufbefehl nach (Bild 11). Bei RST 32 muß zuvor der Funktionskode in das A-Register geladen werden. Weitere Parameter werden in anderen CPU-Registern übergeben.

Bei jedem Eintritt in das Betriebssystem werden alle CPU-Register (außer IY), der aktuelle Inhalt des Busadressenregisters sowie ein spezielles Funktionskodebyte im Stack gerettet. Jede Partition hat ihren eigenen Stack. Die Umschaltung zwischen zwei Partitions erfolgt durch Umladen des IY-Registers und des Stackpointers. Bei der Umschaltung wird der Stackpointer der Partition in den Rettungsbereich der PCA kopiert bzw. von dort entnommen. Das Betriebssystem hat keinen eigenen Stack, sondern benutzt den Stack der jeweiligen Partition. Wenn das Betriebssystem aktiv ist (im sog. Systemzustand des Mikrorechners),

Tafel 4: Übersicht über Systemanweisungen (Auswahl)

Funktion	Parameter	Wirkung
TERMINATE	—	Beenden des Programmablaufs in der aktuellen Partition (Deaktivieren)
SWITCH	—	Zuteilung von Laufzeit an die erste Partition in der Laufzeitwarteschlange
SWITCH TO PARTITION	Partition-Identifizierer	Zuteilung von Laufzeit an die angegebene Partition
SUSPEND PARTITIONS	Partition-Identifizierer (mehrere sind möglich)	Ausschließen der angegebenen Partitions von der weiteren Bearbeitung
SUSPEND EVENTS	ECB-Adressen (mehrere sind möglich)	Ausschließen der angegebenen Ereignisse von der Auslösung
WAKE UP PARTITIONS	Partition-Identifizierer (mehrere sind möglich)	Wiederzulassen der angegebenen Partitions zur weiteren Bearbeitung
WAKE UP EVENTS	ECB-Adressen (mehrere sind möglich)	Wiederzulassen der angegebenen Ereignisse zur Auslösung
CAUSE	ECB-Adresse, bis zu vier Parameter	Auslösen des angegebenen Ereignisses im selben Mikrorechner
INTERRUPT	ECB-Adresse, Interruptvektor, bis zu vier Parameter	Auslösen des angegebenen Ereignisses in einer anderen Einrichtung
WAIT	WAIT-Vektor, ECB-Adresse(n)	Warten bis zum Eintreffen der angegebenen Ereignisse
FREE	ECB-Adresse(n)	Einschalten des Free-Bits in den angegebenen ECBs
BUS CONTROL SERVICE	spezielle Steuerinformation	Erteilen von Aufträgen an den Mikrorechner der Bussteuerung
DELAY	Verzögerungszeit	Verzögerung der Programmbearbeitung in der angegebenen Partition um das angegebene Zeitintervall
WATCHDOG	ECB-Adresse des zu überwachenden Ereignisses, ECB-Adresse des Fehlerereignisses, Zeitintervall	Zeitkontrolle für das zu überwachende Ereignis. Trifft dies nicht im angegebenen Zeitintervall ein, wird das Fehlerereignis ausgelöst.
WATCHDOG OFF	ECB-Adresse des zu überwachenden Ereignisses	Ausschalten der Zeitkontrolle für das zu überwachende Ereignis
TRANSFER	Objektidentifizierer	Übergang (Verzweigung zum angegebenen Programmobjekt)
FETCH	Objektidentifizierer	Laden des angegebenen Programmobjekts in den RAM
GET	Objektidentifizierer	Laden des angegebenen Datenbereichsobjekts in den RAM
PUT	Objektidentifizierer	Schreiben des angegebenen Datenbereichsobjekts auf den externen Datenträger
PRESENCE	Objektidentifizierer	Testen der Anwesenheit des angegebenen Objekts im RAM
PURGE	Transientbereichsadresse	Neu-Initialisieren des Steuerblocks des angegebenen Transientbereiches

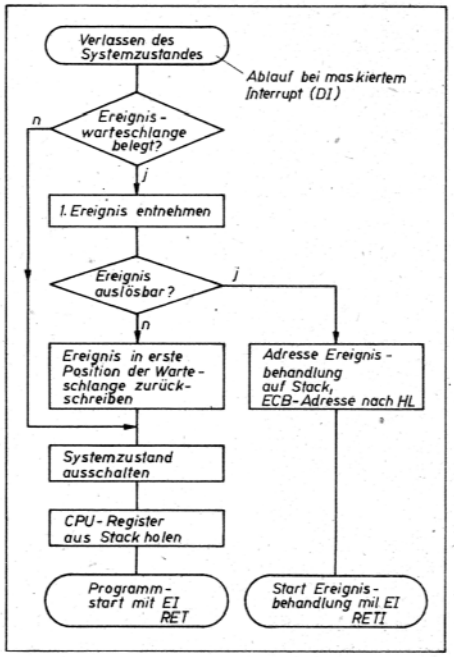

Bild 12: Verlassen des Systemzustandes

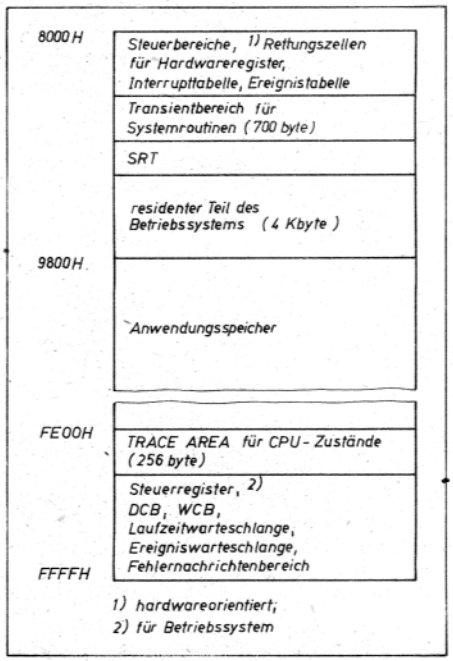

Bild 13: Aufteilung des RAM in einem Mikrorechner

ist der Mikrorechner physisch weiterhin unterbrechbar, um die Bedienung direkt gesteuerter peripherer Einrichtungen weiterhin zu gewährleisten (s. Bild 10). Werden durch Unterbrechungen im Systemzustand logische Ereignisse ausgelöst, so gelangen diese in die Ereigniswarteschlange. Diese wird stets vor dem Verlassen des Systemzustandes inspiziert (Bild 12).

Es ist möglich, Systemaufrufe ineinander zu schachteln. Um bei der Rückkehr aus einer Systemfunktion das Fortsetzen der vorhergehenden zu gewährleisten, wird das erwähnte Funktionskodebyte im Stack mitgeführt. Das Rückkehrprogramm des Betriebssystems analysiert dieses Byte, um erforderlichenfalls die vorhergehende Systemfunktion wieder bereitzustellen.

Jeder Mikrorechner hat eine eigene Kopie des Betriebssystems. Um Speicherplatz zu sparen, sind viele Systemroutinen nur einmal an zentraler Stelle gespeichert. Zur Ausführung werden sie jeweils in einen Transientbereich des betreffenden Mikrorechners transportiert. Die Verbindung zwischen Funktionskode und Startadresse bzw. Position der Systemroutine wird über eine Systemtabelle (System Reference Table, SRT) hergestellt, die analog zur Objekttabelle gemäß Bild 4 aufgebaut ist.
Wird ein Mikrorechner zeitweilig in anderer Weise genutzt, so kann der Speicherplatz des Betriebssystems anderweitig verwendet werden. Bei Rückkehr zum normalen Betrieb ist es unproblematisch, eine Kopie des Betriebssystems aus einem anderen Mikrorechner zu laden.
Tafel 4 gibt einen Überblick über wesentliche Systemfunktionen (Auswahl), im Bild 13 wird die Aufteilung des RAM in einem Mikrorechner veranschaulicht.

Literatur

[1] Schindler, M.: Multiprozessor-Architekturen: Höhere Leistung nur mit neuen Konzepten. Elektronik, München 33 (1984) 17, S. 39–44
[2] Organick, E. I.: Computer System Organization: The B 5700/B 6700 Series. New York, London: Academic Press 1973
[3] Bell, C. G.; Newell, A.: Computer Structures: Readings and Examples. New York: McGraw Hill 1971
[4] Kahn, K. C.: Object-oriented languages tackle massive programming headaches. Electronics, New York 55 (1982) Nov. 17, S. 141–145