Zeitschrift: Bulletin des Schweizerischen Elektrotechnischen Vereins, des Verbandes Schweizerischer Elektrizitätsunternehmen = Bulletin de l'Association suisse des électriciens, de l'Association des entreprises électriques suisses

Herausgeber: Schweizerischer Elektrotechnischer Verein ; Verband Schweizerischer Elektrizitätsunternehmen

Band: 72 (1981)

Heft: 23

Artikel: Echtzeit-Betriebssysteme

Autor: Mühlemann, K.

DOI: https://doi.org/10.5169/seals-905178

Echtzeit-Betriebssysteme

Von K. Mühlemann

681.3.014

Echtzeit-Betriebssysteme teilen die gemeinsamen Betriebsmittel (Prozessoren, Speicher, Peripheriegeräte, Daten, Files, Programme usw.) den Benützern des Computersystems unter Einhaltung zeitlicher Bedingungen zu. Je nach Zielsetzung muss das Betriebssystem optimiert werden können. Dies erschwert die Normungsanstrengungen, welche am ehesten im Bereich des Betriebssystemskerns Erfolgsaussichten haben. Dessen Aufgaben, d.h. die Synchronisation, die Prozessverwaltung, die Ein-/Ausgabe, die Kommunikation und die Ausnahmebehandlung, werden für den allgemeinen Fall eines Computernetzwerkes bestehend aus Multiprozessorknoten dargestellt.

Les systèmes d'exploitation en temps-réel affectent les ressources communes (processeurs, mémoires, périphériques, données, fichiers, programmes, etc.) aux utilisateurs du système d'ordinateur tout en remplissant des conditions temporelles. Le système d'exploitation doit être optimalisable selon le but qui lui est attribué. Cela rend difficile les efforts de normalisation. Ceux-ci ont le plus de chance de réussir dans le domaine du noyau du système d'exploitation. Les tâches de celui-ci, c'est-à-dire la synchronisation, la gestion des processus, des entrées et sorties, la communication et le traitement des exceptions, sont présentées pour le cas général d'un réseau d'ordinateurs composé de nœuds multi-ordinateurs.

1. Einleitung

Unter einem Betriebssystem versteht man jene grundlegenden Softwarekomponenten, die zur effizienten und geordneten Zuteilung der gemeinsamen Betriebsmittel an die Benützer des Computersystems dienen. Das Betriebssystem bildet das Interface zwischen dem Computersystem und dem Benützer.

Man unterscheidet zwischen physikalischen Betriebsmitteln (z.B. Prozessoren, Speichern, Ein-/Ausgabe-Geräten und Verbindungsnetzwerken) und logischen Betriebsmitteln (z.B. Daten, Files und Programmen). Bei der Verwaltung der gemeinsamen Betriebsmittel verfolgen die Betriebssysteme eine Strategie, die auf den Verwendungszweck des Computersystems ausgerichtet ist. Damit soll die Zuteilung jenes Betriebsmittels optimiert werden, das kosten- oder anwendungsbedingt am besten ausgenützt werden muss. In der Vergangeheit war das oft die Zentraleinheit des Computersystems. Seit dank der Mikroprozessortechnik immer leistungsfähigere Zentraleinheiten zu günstigen Preisen verfügbar werden, verschiebt sich der Akzent mehr zu den weiterhin teuren Speicher- und Peripherieeinheiten. In einem Datenbanksystem kann aber durchaus der rasche Zugang zu den Daten gegenüber der Speicher- oder Peripherieausnützung in den Vordergrund rücken.

Bei einem *Echtzeit*-Betriebssystem erwartet man zudem, dass es auf Ereignisse reagieren kann, die in einer bestimmten zeitlichen Beziehung stehen. Insbesondere gilt das für Signale von ausserhalb des Computersystems, die nicht über längere Zeit gültig bleiben. Die computerisierte Messwerterfassung und Regelung liefern typische Beispiele. Echtzeitprobleme treten überdies bei der Ein-/Ausgabe auf. In gewissem Sinne kann man also *alle* Betriebssysteme als Echtzeit-Betriebssysteme auffassen.

Für einen wirksamen Computereinsatz ist ein Betriebssystem unabdingbar. In ihm liegen die Flexibilität und damit die Benützerfreundlichkeit verankert. Eine Normung wäre daher sehr willkommen. Leider existieren, abgesehen von den Interessenskonflikten der Computerhersteller, mehrere Schwierigkeiten, die eine Normung behindern:

a) Die Betriebsmittel, zu deren Verwaltung das Betriebssystem dient, sind in den wenigsten Fällen genormt.

b) Betriebssysteme müssen auf ihre Anwendung zugeschnitten werden.

c) Die Forschung über Betriebssysteme steht noch am Anfang [1].

Eine Vereinheitlichung sollte jedoch bei den *Konzepten* auf der Stufe des Kernbetriebssystems möglich sein. Das Kernbetriebssystem bildet die unterste Schicht des Betriebssystems. Es löst Grundaufgaben und ist damit weitgehend von Strategiefragen unabhängig. In den meisten Fällen hat das Kernbetriebssystem einen relativ geringen Umfang, seine Effizienz ist jedoch entscheidend für die Leistungsfähigkeit des Gesamtsystems. Aus diesem Grund wird das Kernbetriebssystem auch heute noch meist in Assembly-Sprache geschrieben, während die oberen Hierarchiestufen eines Betriebssystems oft in höheren Computersprachen programmiert werden (Tabelle I).

2. Aufgaben eines Kernbetriebssystems

Das Kernbetriebssystem bildet die Grundlage für den Aufbau der höheren Betriebssystemstufen und für die Implementierung höherer Computersprachen. Konkret bedeutet dies die Unterstützung mehrerer Benützer in einem Computersystem. Diese Vervielfachung der Benützer ist wesentlich, um (trotz der durch die mechanischen Peripheriegeräte bedingten Wartezeiten) eine hohe Betriebsmittelausnützung zu gewährleisten. Während nämlich ein Benützer auf eine neue Eingabe wartet, kann durchaus ein anderer Benützer die arithmetischen Fähigkeiten einer Zentraleinheit ausnützen. Im Computerjargon werden die erwähnten Benützer als Tasks oder Prozesse bezeichnet. Die Grundaufgabe des Kernbetriebssystems besteht also in der Unterstützung gleichzeitiger Prozesse.

In der Vergangenheit teilten sich die gleichzeitigen Prozesse meist in eine *einzige* Zentraleinheit. Die Beschränkung auf eine einzige Zentraleinheit ist jedoch künstlich, da eine Zentraleinheit ein Betriebsmittel wie jedes andere ist und vervielfacht

Gegenüberstellung von Kernbetriebssystem und höheren Hierarchiestufen eines Betriebssystems Tabelle I

	Kernbetriebssystem	Höhere Betriebssystemstufen
Aufgaben	Prozeßsynchronisation Prozessverwaltung Ein-/Ausgabe Kommunikation Ausnahmebehandlung	Prozeßsteuerung (Scheduling) Speicherverwaltung Filesystem Datenbanksystem usw.
Programmiersprache	Assembly	höhere Sprache
Charakter	strategieunabhängig	strategieabhängig
Umfang	klein	gross

werden kann. Dass dies selten geschah, hatte mehrere Gründe: Die Kosten der Zentraleinheiten waren hoch, und die Verwaltung mehrerer Zentraleinheiten im Betriebssystem bot grosse Schwierigkeiten. Im weiteren war es infolge des technischen Fortschritts möglich, mit geringem Aufwand erhebliche Leistungssteigerungen mit einer einzigen Zentraleinheit zu realisieren. Heute stehen jedoch leistungsfähige und preisgünstige Mikroprozessoren als Zentraleinheiten zur Verfügung, und Leistungssteigerungen einzelner Zentraleinheiten erfordern einen unverhältnismässigen Aufwand. Durch Multiprozessoren sind deshalb Vorteile wie höhere Leistungen, erhöhte Zuverlässigkeit, grössere Verfügbarkeit und modulares Systemwachstum absehbar geworden.

Damit erhielt auch die Arbeit an den Multiprozessor-Betriebssystemen neuen Auftrieb. Eine von der Europäischen Gemeinschaft geförderte Anstrengung findet im TC 8 on Real Time Operating Systems des European Workshop on Industrial Computer Systems (EWICS) statt. Es ist das Ziel dieses Komitees, Richtlinien und systemunabhängige Konzepte zu entwickeln, um beim Aufbau von Echtzeit-Betriebssystemen maximale Zuverlässigkeit und Übertragbarkeit zu erzielen [2]. Die bisherige Arbeit konzentrierte sich auf das Kernbetriebssystem, das inzwischen weitgehend abgeschlossen ist. Bei der Behandlung wurde konsequent der Multiprozessorfall angenommen, aus dem der Einprozessorfall durch Spezialisierung unmittelbar hergeleitet werden kann.

Im folgenden wird eine Übersicht über die erzielten Resultate vermittelt. Zunächst befasst sich die Darstellung mit dem eng gekoppelten Multiprozessor, in dem die Prozesse über gemeinsame Variable in einem gemeinsamen Speicher kooperieren. Darauf folgt die Verallgemeinerung auf ein lose gekoppeltes Netzwerk, in dem die Prozesse über Datenkanäle kommunizieren. Da die Netzwerkknoten aus eng gekoppelten Multiprozessoren bestehen können, gelten die Ergebnisse für den offenbar allgemeinsten Fall eines Computersystems. Die in der Praxis wichtigen Spezialfälle, nämlich das Netzwerk, bestehend aus Einprozessorsystemen, das Multiprozessorsystem und das Einprozessorsystem sind dabei also gleich mitbehandelt.

2.1 Prozeßsynchronisation

Die Zusammenarbeit zwischen gleichzeitigen Prozessen beruht auf einem gegenseitigen Informationsaustausch. Dabei müssen gewisse zeitliche Einschränkungen eingehalten werden. Die Massnahmen zur Realisierung dieser Einschränkungen werden als Prozeßsynchronisation bezeichnet. Zwei typische Synchronisationsarten sind das Produzenten/Konsumenten-Verhältnis und der gegenseitige Ausschluss.

Beim Produzenten/Konsumenten-Verhältnis erzeugt ein Produzentenprozess Information, die von einem Konsumentenprozess weiterverwendet wird. Dies ist aber erst nach Abschluss der Informationsproduktion möglich. Der Konsument muss so lange warten.

Ein gegenseitiger Ausschluss ist nötig, wenn sich mehrere Prozesse über eine Anzahl gemeinsamer Variablen verständigen. Während die gemeinsamen Variablen von einem Prozess verändert werden, können sie sich kurzzeitig in einem nicht konsistenten Übergangszustand befinden. Greift zu diesem Zeitpunkt ein anderer Prozess zu den gemeinsamen Variablen zu, so könnte er falsche Schlüsse ziehen. Man muss deshalb dafür sorgen, dass nur solche Prozesse gleichzeitig zu den gemeinsamen Variablen zugreifen, die dadurch nicht irregeleitet werden können. Die anderen Prozesse müssen temporär vom Zugriff ausgeschlossen werden.

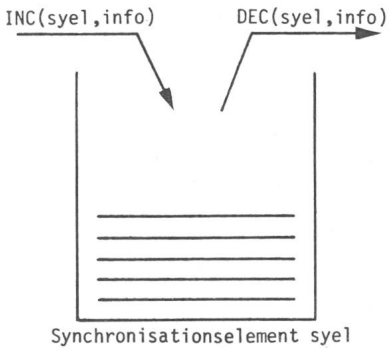

Fig. 1 Synchronisationselement und Synchronisationsoperationen

Für die Behandlung der Synchronisation auf der Kernbetriebssystemstufe bestehen eine ganze Reihe von Vorschlägen. Diese eignen sich entweder besonders gut für das Produzenten/Konsumenten-Verhältnis, wie etwa die Meldungssynchronisation [3; 4], oder aber für den gegenseitigen Ausschluss, wie etwa die Semaphore [5; 6]. Das TC 8 on Real Time Operating Systems schlägt hingegen ein Synchronisationssystem vor, das die Vorteile der Meldungs- und der Semaphorsynchronisation vereinigt. Dieser Vorschlag beruht auf einer Art Mailbox, welche Synchronisationselement genannt wird [2; 7]. Die Synchronisationsoperationen werden aus historischen Gründen mit INC und DEC bezeichnet.

Die Operation INC (syel, info) übergibt das Informationselement info an das Synchronisationselement syel. Die entsprechende Operation DEC (syel, info) fordert ein Informationselement info vom Synchronisationselement syel an (Fig. 1). Liegt wenigstens ein Informationselement vor, so wird es aus dem Synchronisationselement herausgelöst, und der anfordernde Prozess fährt fort. Ist das Synchronisationselement jedoch leer, so erfolgt eine Prozessumschaltung. Der anfordernde Prozess gibt seinen Prozessor zugunsten eines andern Prozesses auf, der zum Fortfahren bereit ist. Die Operation DEC wird dann zu einem späteren Zeitpunkt vollendet, wenn nämlich das erforderliche Informationselement verfügbar wird.

Mittels eines Synchronisationselementes kann unmittelbar ein Produzenten/Konsumenten-Verhältnis mit einem unendlichen Meldungspuffer realisiert werden (Fig. 2). In dieser Hinsicht ist INC/DEC also der Meldungssynchronisation ähnlich.

Wenn ein Synchronisationselement mit «dummy»-Information operiert, so entspricht es einem Semaphor. Eine kritische Sektion beispielsweise, bei der ein Prozess alle andern ausschliesst, kann mit einem Synchronisationselement mutex realisiert werden, das mit einem dummy-Informationselement initialisiert ist. Das dummy-Informationselement hat dabei

Produzent	*Konsument*
repeat	*repeat*
produce (info);	DEC (buf, info);
INC (buf, info)	consume (info)
forever	*forever*

Fig. 2 Produzenten/Konsumenten-Verhältnis realisiert mit INC/DEC

die Bedeutung eines Passierscheins, der dem Prozess, der ihn besitzt, den Zugriff zu den gemeinsamen Variablen erlaubt (Fig. 3).

Wie an anderer Stelle [8] gezeigt wurde, erlaubt INC/DEC alle bekannten Synchronisationssysteme nachzubilden. Insbesondere gilt das auch für die strukturierten Synchronisationssysteme höherer Sprachen, wie Monitore [9], kritische Regionen, bedingte kritische Regionen und Events [4]. Kürzlich wurde auch gezeigt [10], dass INC/DEC sich zur Realisierung der mächtigen Ada-Rendez-vous-Synchronisation [11] eignet. Damit genügt INC/DEC offenbar der Anforderung an das Kernbetriebssystem, als Grundlage für die Realisierung höherer Programmiersprachen zu dienen.

```
.
.
.
DEC (mutex, dummy);
critical section;
INC (mutex, dummy);
.
.
.
```

Fig. 3 Kritische Sektion realisiert mit INC/DEC

2.2 Prozessverwaltung

In der Darstellung der Synchronisation wurden impliziert drei Prozesszustände unterschieden, nämlich laufend, blokkiert und bereit. Ein *laufender* Prozess verfügt über einen Prozessor und macht Verarbeitungsfortschritte. In einem Multiprozessorsystem mit N Prozessoren können maximal N Prozesse laufend sein. Ein *blockierter* Prozess kann aus Gründen der Synchronisation nicht fortfahren. Einem *bereiten* Prozess fehlt einzig der Prozessor, um mit der Verarbeitung fortfahren zu können. In Figur 4 sind diese Prozesszustände und die Übergänge zwischen ihnen dargestellt.

Für gewisse einfache Betriebssysteme genügen diese drei Prozesszustände. Die vorliegenden Prozesse besitzen dann *statischen* Charakter, d.h. während der ganzen Systemlebensdauer sind dieselben Prozesse vorhanden.

Soll das Computersystem jedoch *dynamisch* verwaltet werden, so muss es möglich sein, neue Prozesse einzuführen oder bisherige Prozesse zu entfernen. Dies wird als Prozessverwaltung bezeichnet. Eine besonders wichtige Aufgabe der Prozessverwaltung besteht darin, einen bereits bestehenden Prozess zeitweilig von der Bearbeitung auszuschliessen. Da er in den aktiven Prozesszuständen (laufend, blockiert und bereit) entweder bereits läuft oder jederzeit zum Laufen kommen kann, muss er aus den aktiven Prozesszuständen entfernt werden. Diese Operation heisst DEACTIVATE. Sie überträgt einen spezifizierten aktiven Prozess in den inaktiven Prozesszustand. Der Prozess ist dadurch der kurzfristigen Prozeßsteuerung durch die Synchronisationsoperationen entzogen. Er kann zu einem späteren Zeitpunkt durch ACTIVATE wieder aktiviert werden. Die Operationen DEACTIVATE und ACTIVATE sind erforderlich, um *Speicherverwaltung* und *Ausnahmebehandlung* auf höheren Betriebssystemstufen zu realisieren.

Die Übergänge zwischen dem inaktiven und dem undefinierten Prozesszustand dienen zur endgültigen Vernichtung bestehender Prozesse bzw. zur Erzeugung neuer Prozesse. Die entsprechenden Operationen sind DELETE und CREATE.

Der Vollständigkeit halber seien noch die Operationen RETIRE und TERMINATE erwähnt, mittels derer sich ein laufender Prozess selbst deaktivieren bzw. vernichten kann.

2.3 Ein-/Ausgabe

Unter Ein-/Ausgabe versteht man den Transport von Information über die Grenze des Computersystems. Bei der Eingabe erfolgt der Informationstransport von einer Informationsquelle zum Computersystem. Die Ausgabe bezeichnet umgekehrt den Informationstransport vom Computersystem zu einer Informationssenke.

Die Ein-/Ausgabe liesse sich ideal lösen, wenn die Peripheriegeräte als Prozessoren des Computersystems aufgefasst werden könnten. Diese Prozessoren würden einerseits den Informationstransport durchführen und könnten andererseits die Operationen des Kernbetriebssystems ausführen. Die Ein-/Ausgabe-Aufträge könnten also mittels der besprochenen Synchronisationsoperationen an die Ein-/Ausgabe-Prozesse übergeben werden, die auf den Peripheriegeräte-Prozessoren laufen.

Aus Mangel an Normung bei Bussen, Schnittstellen und Betriebssystemen ist man bis heute leider noch nicht so weit. Deshalb ist man kurzfristig gezwungen, das angestrebte Verhalten mit Hilfe der Zentraleinheiten des Computersystems zu *simulieren*. Die mangelnden Datenverarbeitungsfähigkeiten der Peripheriegeräte äussern sich dann in Form von Interruptroutinen. Diese kann man als Prozessorleistung interpretieren, welche die Peripheriegeräte von den Zentraleinheiten ausleihen. Figur 5 stellt die grundsätzliche Ein-/Ausgabe-Struktur dar. Diese gilt sowohl für die Ein- wie auch für die Ausgabe. Der Benützerprozess teilt dem Ein-/Ausgabe-Prozess mittels INC seinen Ein-/Ausgabe-Auftrag mit. Dabei bezeichnet rtnsyel ein Synchronisationselement, auf dem die Beendigung der Ein-/Ausgabe gemeldet werden soll.

Wenn der Ein-/Ausgabe-Prozess einen Auftrag erhalten hat, startet er eine Interruptroutine, welche den Transfer selbständig, im Gegentakt mit der Hardware durchführt. Die Beendigung des Transfers wird dem Ein-/Ausgabe-Prozess mittels INC (devint, dummy) auf dem Synchronisationselement devint gemeldet. Darauf können die Antwortparameter über das

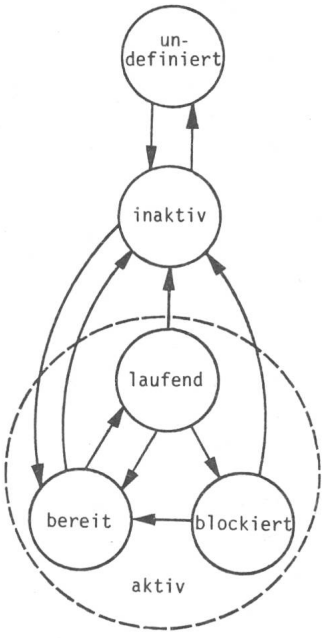

Fig. 4 Prozesszustände und Zustandsänderungen

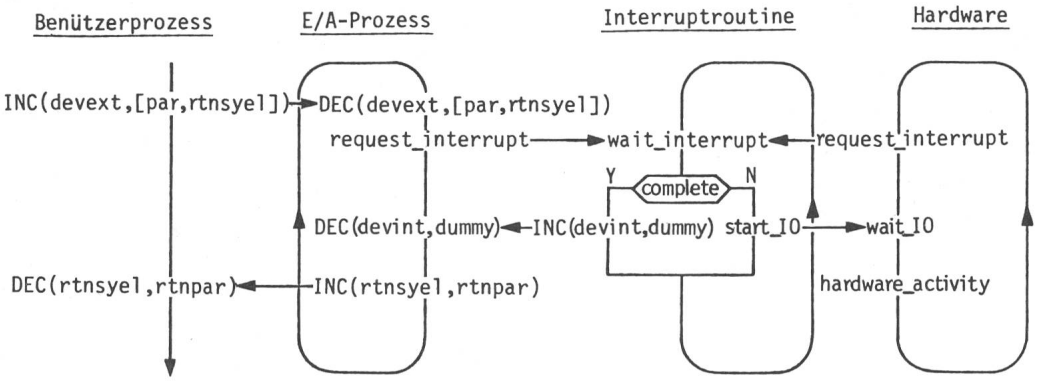

Fig. 5
Ein-/Ausgabe-Struktur

Synchronisationselement rtnsyel an den Benützerprozess übergeben werden.

2.4 Notifikation

Ein heute noch kaum bearbeitetes Gebiet ist die Behandlung von Interrupts in einem Multiprozessorsystem. Als Beispiel diene die Ein-/Ausgabe des vorhergehenden Abschnittes. Trifft ein Geräteinterrupt ein, so sollte man für dessen Behandlung einen Prozessor auswählen, der auf möglichst tiefer Priorität arbeitet oder eventuell gar nichts zu tun hat. Da sich in einem Multiprozessorsystem die Situation laufend verändert, kann dieser geeignetste Prozessor zwischen zwei aufeinanderfolgenden Interrupts ändern. Es sollte also möglich sein, die Interrupts dynamisch umzuleiten.

Ein ähnlich gelagertes Problem stellt die Prozessverdrängung (Preemption) dar. Eine Prozessverdrängung sollte stattfinden, wenn durch INC oder ACTIVATE ein Prozess bereit wird, der höhere Priorität besitzt als der mit der momentan tiefsten Priorität laufende Prozess. Die beiden Prozesse vertauschen dann ihre Rollen.

Für die Prozessverdrängung ist jedoch die Mitarbeit des Prozessors nötig, auf dem der zu verdrängende Prozess läuft. Dieser Prozessor muss ja den Context des alten Prozesses retten, damit dieser zu einem späteren Zeitpunkt fortfahren kann. Man muss also die Aufmerksamkeit des betreffenden Prozessors gewinnen. Dies kann nur über das Interruptsystem geschehen; es stellt die einzige Möglichkeit dar, asynchron in die Prozessausführung einzugreifen. Zur Unterscheidung von den Geräteinterrupts wurde diese Art von Interrupt vom TC 8 on Real Time Operating Systems *Notifikation* getauft. Auch hier ist zu beachten, dass der Notifikationsstimulus dynamisch zum Zielprozessor geleitet werden muss, da sich die Situation im Multiprozessorsystem fortlaufend ändert.

Eine weitere Art von Interprozessornotifikation ist erforderlich, um einen laufenden Prozess zu deaktivieren. Auch in diesem Fall wird die Aufmerksamkeit des ausführenden Prozessors durch den Notifikationsstimulus erzwungen. Der Prozessor deaktiviert darauf den auf ihm laufenden Prozess. Die Notifikation muss berücksichtigen, dass in einem Multiprozessorsystem die Identität des ausführenden Prozessors dynamisch ändern kann.

Die dynamische Zuteilung der Notifikationsstimuli muss zudem noch Konflikte zwischen mehreren gleichzeitig anstehenden Notifikationsaufträgen lösen. Reine Softwarelösungen sind zuwenig effizient. Vielmehr sollten die hardwarenahen Interrupts und Notifikationen auch mit einer Hardwarelösung behandelt werden. Ein Vorschlag [8; 12] verwendet eine mikroprogrammierte Notifikationseinheit für die Zuteilung der Notifikationen. Neue Entwicklungen auf dem Multiprozessorgebiet sind sich der Notifikationsfragen bewusst. So sieht der Entwurf zum P896 Busstandard [13] Mittel zur Realisierung der Notifikation vor. Die Zuteilung von Notifikationsstimuli ist eindeutig ein neues Erfordernis der Multiprozessoren. In einem Einprozessorsystem entfällt die Prozessorauswahl, da ja nur eine einzige Zentraleinheit existiert.

2.5 Ausnahmebehandlung

Unter einer Ausnahme versteht man eine Abweichung vom normalen Ablauf eines Prozesses. Die Annahme kann die Folge eines Fehlers in der Hardware oder der Software sein, beispielsweise ein Bitfehler im Speicher oder eine Division durch null. Die Ausnahme selbst ist aber kein Fehler. Sie stellt vielmehr ein erwartetes Verhalten dar und wurde deshalb im Computersystem auch entdeckt.

Das Kernbetriebssystem muss Mittel zur Verfügung stellen, um eine Ausnahme behandeln zu können. Es geht dabei darum, den richtigen «Exception Handler» zur Ausführung zu bringen. Die Identität des richtigen Exception Handlers hängt von der Art der Ausnahme, ihrer Entdeckung und ihrer Ursache ab. Ist die Ausnahmeursache ein Fehlverhalten der Rechnerhardware, so wird im allgemeinen das Betriebssystem die nötige Rekonfiguration des Computersystems vornehmen, den Fehler beheben und die Instandsetzungsaktionen anfordern.

Handelt es sich jedoch um ein Fehlverhalten der Anwendungssoftware, so wird ein geeigneter Exception Handler aus der Anwendungssoftware eingesetzt werden. Höhere Programmiersprachen wie Ada [11] sehen vor, dass eine Ausnahme stufenweise in der Anwendungssoftware nach oben gereicht wird, bis ein geeigneter Handler gefunden wird.

Diese Methode ist erfolgreich, wenn die Entdeckung der Ausnahme in demselben Prozess erfolgt, der die Ausnahme verursacht hat. Ist jedoch die Ausnahme von einem andern Prozess verursacht worden, so sollte sie auch in jenem Prozess behandelt werden. Es muss deshalb eine Möglichkeit existieren, eine Ausnahmebedingung an einen andern Prozess weiterzuleiten. Ada sieht zu diesem Zweck die Ausnahme FAILURE vor, die explizit in einem andern Prozess angezeigt werden kann. In einem Einprozessorsystem ist dies einfach zu lösen, da der betreffende Prozess ja sicher nicht läuft, während ein anderer Prozess die Ausnahme entdeckt. In einem Multiprozessorsystem kann der betreffende Prozess jedoch laufen. Im Notifikationsmechanismus sind Mittel vorzusehen, die den laufenden Prozess auf die Ausnahme aufmerksam machen.

3. Netzwerkbetriebssystem

In einem lose gekoppelten Netzwerk kann die Kooperation zwischen den einzelnen Prozessen nicht mit den beschriebenen Synchronisations- und Prozessverwaltungsoperationen organisiert werden. Diese setzen nämlich den Zugang zu den Prozessdeskriptoren und den Synchronisationselementen in einem gemeinsamen Speicher voraus. Ein systemweiter gemeinsamer Speicher existiert jedoch in einem Netzwerk nicht mehr. Trotzdem wäre es vorteilhaft, netzwerkweit mit den bereits beschriebenen Operationen arbeiten zu können.

Diese Möglichkeit bietet sich mit einer Verallgemeinerung der Kernbetriebssystemoperationen [8; 14; 15]. Diese werden zur Unterscheidung durch die Vorsilbe GEN- ergänzt. Anstelle von INC, DEC, ACTIVATE, DEACTIVATE usw. arbeitet man dann mit GENINC, GENDEC, GENACTIVATE, GENDEACTIVATE usw. Diese Operationen besitzen die in Figur 6 dargestellte Struktur. Bei ihrer Ausführung wird zunächst entschieden, ob das anzusprechende Objekt sich im eigenen Netzwerkknoten befindet. In diesem Fall kann direkt die Originaloperation eingesetzt werden, also INC, DEC, ACTIVATE, DEACTIVATE usw. Andernfalls wird der Auftrag über ein *lokal* verfügbares Synchronisationselement (dis) an das Kommunikationssystem weitergeleitet. In den meisten Fällen ist die Angelegenheit für den ausführenden Prozess damit erledigt (Fig. 6a). Einzig in den Fällen, in denen ein Antwortparameter erwartet wird, folgt noch eine DEC-Operation auf ein lokales, für jeden Prozess existierendes Synchronisationselement (priv) (Fig. 6b). Wenn die Operation abgeschlossen ist, kann der aufrufende Prozess die Antwortparameter lokal auf dem Synchronisationselement priv entgegennehmen.

Wenn das Kommunikationssystem den Auftrag vom lokalen Synchronisationselement dis übernommen hat, durchläuft dieser die verschiedenen Schichten des Kommunikationssystems nach den bekannten Methoden [16]. Fragen des Routing und der Flowcontrol werden entschieden. Schliesslich erfolgt die Übermittlung des Auftrages mittels Ein-/Ausgabe über die vorhandenen Kommunikationskanäle. Dabei werden eventuell mehrere Netzwerkknoten durchquert. Am Bestimmungsort liefert das Kommunikationssystem die Aufträge an das Kernbetriebssystem ab. *Stellvertreterprozesse* führen dann die übermittelten Aufträge aus.

In dieser Weise kann das Kernbetriebssystem auf ein Netzwerk ausgedehnt werden. Man beachte, dass die erwähnten Methoden keine Voraussetzungen über die Netzwerkstruktur oder die Art der Vermittlung (Linien-, Meldungs- oder Paketvermittlung) treffen. Zudem ist die Transparenz der Kommunikation gewährleistet. Der Auftraggeberprozess spezifiziert die angesprochenen Objekte (Synchronisationselemente und Prozesse) per Namen und unabhängig von ihrem Aufenthaltsort im Netzwerk. Damit entspricht der Vorschlag den Anforderungen, die an ein modernes Netzwerkbetriebssystem gestellt werden.

4. Schlussfolgerungen

Betriebssysteme müssen auf ihren Verwendungszweck zugeschnitten werden und widersetzen sich deshalb einer Normung. Die Optimierung betrifft jedoch nur die obersten Schichten des Betriebssystems. Prinzipiell wäre das Kernbetriebssystem, welches Grundaufgaben erfüllt, also normbar. Die Normung des Kernbetriebssystems scheitert heute jedoch an

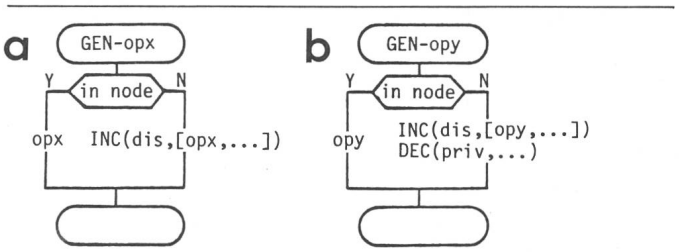

Fig. 6 Struktur der verallgemeinerten Kernbetriebssystemoperationen im Netzwerk
opx = INC, ACTIVATE, DEACTIVATE, DELETE
opy = DEC, CREATE

der fehlenden Normung bei den Betriebsmitteln und an den bis vor kurzem fehlenden Kenntnissen über Multiprozessorsysteme. In diesem Artikel werden die Konzepte des vom TC 8 on Real Time Operating Systems entwickelten Kernbetriebssystems dargelegt. Es ist konsequent auf den Multiprozessorfall ausgerichtet und kann auf ein Netzwerkbetriebssystem verallgemeinert werden. Zwar ist aus den erwähnten Gründen bisher keine Normung erfolgt, doch erlaubt das Betriebssystem zumindest eine Übertragbarkeit der Konzepte. Längerfristig ist eine Realisierung des Kernbetriebssystems auf dem Silizium der Prozessorchips vorzusehen. Bei der Prozessorinitialisierung kann dann die Systemart, Einprozessor, Multiprozessor und/oder Netzwerk, konfiguriert werden. Dadurch kann die Effizienz je nach Systemart optimiert werden, ohne die Bedeutung der Kernbetriebssystemoperationen zu verändern.

Literatur

[1] *D. A. Anderson:* Operating systems. Computer 14(1981)6, p. 69...82.
[2] Technical committee on real time operating systems: Up to date report. European Workshop on Industrial Computer Systems, September 1978; paper OP/SYS I–1–7.
[3] *P. Brinch Hansen:* The nucleus of a multiprogramming system. Communications of the Association for Computing Machinery 13(1970)4, p. 238...241, 250.
[4] *P. Brinch Hansen:* Operating system principles. Englewood Cliffs, N.J., Prentice-Hall, 1973.
[5] *E. W. Dijkstra:* Co-operating sequential processes. Out of 'Programming languages', lectures given in Villard-de-Lans, edited by F. Genuys. London & New York, Academic Press, 1968, p. 43...112.
[6] *E. W. Dijkstra:* The structure of the 'THE'-multiprogramming system. Communications of the Association for Computing Machinery 11(1968)5, p. 341...346.
[7] *G. Schrott:* Common elementary synchronisation functions for an operating system kernel. European Workshop on Industrial Computer Systems, January 1976; paper OP/SYS II–3–1.
[8] *K. Mühlemann:* Ein Beitrag zur Synchronisation in Mehrprozessorsystemen und Computernetzwerken. Dissertation der Eidgenössischen Technischen Hochschule Zürich Nr. 6520, 1980.
[9] *C. A. R. Hoare:* Monitors: an operating system structuring concept. Communications of the Association for Computing Machinery 17(1974)10, p. 549...557.
[10] *K. Mühlemann:* Towards an implementation of Ada rendezvous synchronization. Euromicro Symposium on Microprocessing and Microprogramming, Paris, 1981. Implementing functions: microprocessors and firmware. Amsterdam a.o., North-Holland Publishing Company, p. 289...302.
[11] Reference manual for the Ada programming language. United States Department of Defense, proposed standard document, July 1980.
[12] *K. Mühlemann:* Towards interrupt assignment for multiple processors. Euromicro Symposium on Microprocessing and Microprogramming, London, 1980. Microprocessor systems. Amsterdam a.o., North-Holland Publishing Company, p. 157...166.
[13] *J. D. Nicoud:* Bus normalisés pour microprocesseurs. Bull. SEV/VSE 72(1981) 23, p. 1231...1234.
[14] *K. Mühlemann:* Communication between processors without shared memory. Joint Meeting ACM/IEEE on Interconnected Small Processors, Bern, November 21st, 1978.
[15] *K. Mühlemann:* Towards a model of communication. European Workshop on Industrial Computer Systems, April 1979; paper OP/SYS II–14–1.
[16] ISO/TC 97/SC: Reference model of open systems interconnection. N 227, June 1979.

Adresse des Autors

Dr. *K. Mühlemann*, ASE-CSEE (Association Suisse des Electriciens – Centre Suisse d'Essais des Composants Electroniques), ruelle Vaucher 22, 2000 Neuchâtel.