DEUTSCHES ELEKTRONEN-SYNCHROTRON DESY

DESY DV-71/1
November 1971

Multi-Tasking auf kleinen Rechnern

von

Eckart Raubold

2 HAMBURG 52 · NOTKESTIEG 1

To be sure that your preprints are promptly included in the HIGH ENERGY PHYSICS INDEX, send them to the following address (if possible by air mail):

DESY
Bibliothek
2 Hamburg 52
Notkestieg 1
Germany

Multi - Tasking auf kleinen Rechnern

von

Eckart Raubold

DEUTSCHES ELEKTRONEN-SYNCHROTRON DESY, HAMBURG

Manuskript zu einem Vortrag, gehalten im Juni 1971
in Birlinghoven.

INHALT:

ABSTRACT

Starting from a definition of what multi-tasking is and from an example to illustrate its use and necessity it is tried to introduce a more general look at organisational problems within computers. This generalisation allows to describe what a multi-task "supervisor" in principle should do, what has been achieved in present day systems and what especially can be done in small computer systems. The task supervisor used with the PDP-8's at DESY is given as an example.

1. Was ist Multi - Tasking ?

Unter Multi - Tasking versteht man die zeitlich verzahnte Bearbeitung mehrerer Aufgaben durch einen Rechner. Durch die zeitliche Verzahnung der Bearbeitung entsteht dabei der Eindruck, der Rechner bearbeite mehrere Aufgaben gleichzeitig. Multi - Tasking ist im Gegensatz zu sehen zu Multi-Processing, wo tatsächlich mehrere gekoppelte Rechner parallel an mehreren Aufgaben tätig sind.

Kennzeichnend für Multi - Tasking ist, daß

1. die zu erledigenden Aufgaben nach Prioritäten geordnet sind,
2. Aufgaben niedrigerer Priorität den Rechnerkern freigeben ("unterbrochen" werden), wenn eine Aufgabe höherer Priorität den Rechner benötigt,
3. Aufgaben höherer Priorität, die aus irgendwelchen Gründen im Augenblick nicht weitergeführt werden können, den Rechnerkern freigeben für die Benutzung durch Aufgaben niedrigerer Priorität.

Steht ein Rechner in aktiver Wechselbeziehung zu seiner Umwelt (E/A Geräte, Meßinstrumente, Stellglieder o.ä.), so ist Multi-Tasking in den meisten Fällen die Voraussetzung für eine optimale Nutzung der Anlage; bei Prozeßsteuerungs-Anwendung (Real-Zeit-Anwendung) ist Multi-Tasking sogar oft unumgänglich.

Beispiel: Eine Folge von externen Signalen mit Signalabstand t_s muß vom Rechner behandelt werden. Die folgenden Aufgaben sind zu erledigen:

1. für jedes Signal Einlesen eines Meßwertes und Vergleich mit vorgegebenen Grenzwerten: Zeitbedarf = t_1.
2. für je 10 angesammelte Meßwerte Approximation der zeitlichen Entwicklung der Meßwerte durch eine Gerade und Prüfen auf drohende Grenzwertüberschreitung durch Extrapolation; Zeitbedarf=t_2.

3. für je 100 angesammelte Meßwerte
Anpassen einer mehrparametrigen Funktion
für die Prozeßoptimalisierung; Zeitbedarf= t_3.

Angenommen es gelte

$$t_s > t_1 + \frac{t_2}{10} + \frac{t_3}{100} \qquad (1)$$

und

$$t_3 > t_2 > t_s \qquad (2)$$

Aus (1) folgt, daß der Rechner die geforderte Aufgabe bewältigen kann. Wegen (2) kann die Aufgabe aber nicht durch serielles Ausführen der Teilaufgaben erledigt werden. Die drei Teilaufgaben müssen auf drei parallele Tasks verteilt werden, und zwar so, daß die Aufgabe 1. die höchste, Aufgabe 2. die mittlere und Aufgabe 3. die niedrigste Priorität hat.

Das Beispiel demonstriert gleichzeitig, daß für Multi-Tasking i.A. auch die Möglichkeit für das Starten einer Task auf Grund einer rechnerinternen Bedingung erforderlich ist:
Die Teilaufgaben 2. und 3. müssen abhängig von einem rechnerinternen Zählerstand gestartet werden.

Im vergangenen war noch nicht auf die mit Multi-Tasking verbundenen organisatorischen Probleme eingegangen worden. Im folgenden sollen zunächst durch Abstraktion von der rechnergebundenen Vorstellung eine Klärung der relevanten Begriffe durchgeführt, dann allgemeine Forderungen für ein Multi-Tasking-Verwaltungssystem formuliert und schließlich die für kleine Rechner akzeptablen Kompromisse und deren Realisierung diskutiert werden.

2. Abstraktion vom Rechner

Von Dijkstra [1] ist der Begriff der "harmonisch kooperierenden sequentiellen Prozesse" geprägt worden. Dahinter steht die Vorstellung von einer Reihe seitlich parallel ablaufender Vorgänge, die, bis auf wenige, wohl definierte Verkoppelungen, von einander unabhängig fortschreiten. Dieses Modell läßt sich wie folgt präzisieren:

Die Prozesse finden statt über einer endlichen Menge von Hilfsmitteln, die die Prozesse für ihr Fortschreiten benötigen und deren Zustand den Zustand jedes einzelnen Prozesses beschreibt.

Die Prozesse sind verkoppelt einmal durch ihre Konkurrenz um die endliche Zahl vorhandener Hilfsmittel (implizite Abhängigkeit), zum anderen dadurch, daß sie sich wechselseitig durch das Erzeugen gewisser Zustände von Hilfsmitteln bzw. durch warten auf das Eintreten dieser Zustände bedingen (explizite Abhängigkeit).

Die oben genannte Voraussetzung über "wenige, wohldefinierte Verkopplungen" bedeutet, daß jeder Prozeß in Prozeßstücke zerfällt, die völlig unabhängig von der Prozeßumgebung ablaufen. Die Grenze zwischen zwei Prozeßstücken markiert eine Stelle, wo eine oder mehrere der folgenden Bedingungen eintreten:

1. Der Prozeß benötigt für seine Fortsetzung weitere freie Hilfsmittel.
2. Der Prozeß gibt Hilfsmittel frei, die er nicht mehr benötigt.
3. Der Prozeß kann erst fortgesetzt werden, wenn gewisse Hilfsmittel sich in einem bestimmten Zustand befinden.
4. Der Prozeß hat gewisse Hilfsmittel in einen Zustand versetzt, der von anderen Prozessen erwartet wird.

Eine spezielle Teilmenge der Menge der Hilfsmittel sind die Prozessoren. Diese sind die Träger der Prozesse in zweierlei Hinsicht:

1. Jeder Prozeß benötigt für sein Fortschreiten mindestens einen Prozessor.
2. Der Zustand des zu einem Prozeß gehörigen Prozessors definiert am Ende eines Prozeßstückes eindeutig das nachfolgende Prozeßstück.

Die Prozessoren gestatten also erst die eindeutige Aneinanderreihung verschiedener Prozeßstücke zu Prozessen. Die Zahl der Prozessoren gibt die maximale Zahl der zeitlich simultan ablaufenden Prozesse an.

Hilfsmittel oder Kombinationen von Hilfsmitteln (hier auch kurz als Hilfsmittel bezeichnet) können die folgenden Eigenschaften haben:

1. a) Entweder: Durch Benutzung wird der Zustand des Hilfsmittels immer zerstört (nicht wieder benutzbar).

   b) Oder : Durch Benutzung wird der Zustand nur zerstört, wenn der Benutzer das beabsichtigt (wieder benutzbar).

2. a) Entweder: Der Zustand des Hilfsmittels ist auslesbar und nach Belieben wiederherstellbar (unterbrechbar).

   b) Oder : Der Zustand des Hilfsmittels ist nicht auslesbar oder nicht nach Belieben wiederherstellbar (ununterbrechbar).

3. a) Entweder: Bei zerstörungsfreier Benutzung kann das Hilfsmittel gleichzeitig von mehreren Prozessen benutzt werden (parallel benutzbar).

   b) Oder : Selbst bei zerstörungsfreier Benutzung kann das Hilfsmittel nicht gleichzeitig von mehreren Prozessen benutzt werden (seriell benutzbar).

Die Eigenschaften 3. sind nur bei Wiederbenutzbarkeit (Eigenschaft 1.b) möglich.

Ein Prozeß kann sich in Bezug auf Hilfsmittel wie folgt verhalten:

1. Bei der Anforderung:

   a) zustandsbezogen, d.h. Hilfsmittel mit einem bestimmten Zustand werden verlangt

   b) zustandsfrei, d.h. freie Hilfsmittel unabhängig von deren gegenwärtigem Zustand werden verlangt.

2. Bei der Benutzung:

   a) zustandsändernd, d.h. der Prozeß verändert den Hilfsmittelzustand absichtlich.

b) zustandserhaltend.

3. Bei der Freigabe:

a) zustandsbezogen, d.h. der augenblicklich vorliegende Zustand ist für späteren Gebrauch zu erhalten.

b) zustandsfrei, d.h. Freigabe für beliebige weitere Benutzung.

Sind einige der im betrachteten System vorhandenen Hilfsmittel unterbrechbar (Eigenschaft 2.a), so kann diese Eigenschaft wie folgt ausgenutzt werden:

Ein Prozeß, der ein solches Hilfsmittel dringend benötigt, es aber besetzt vorfindet, rettet den Zustand des Hilfsmittels in ein Speichermedium, benutzt das Hilfsmittel (wobei dessen Zustand sich ändern kann), und restauriert den Zustand des Hilfsmittels vor der Wiederfreigabe.

Bei der Entscheidung, ob ein Hilfsmittel in dem beschriebenen Sinn "unterbrochen" werden sollte oder nicht, können zwei Gesichtspunkte eine Rolle spielen:

1. Die relative Wichtigkeit des anfordernden und des besitzenden Prozesses (Priorität).

2. Ob der besitzende Prozeß das Hilfsmittel tatsächlich im Augenblick benutzt; er könnte sich ja im Wartezustand auf andere Hilfsmittel befinden und daher das benötigte Hilfsmittel ungenutzt blockieren.

An dieser Stelle kann nun formuliert werden, was Multi-Tasking in dem geschilderten Modell bedeutet: Ist ein Prozessor ein unterbrechbares Hilfsmittel und wird nach Bedarf jeweils einem von mehreren Prozessen zugeteilt, so betreibt der Prozessor Multi-Tasking.

3. Forderungen an eine Multi-Task-Verwaltung

Ich wende mich jetzt wieder dem Spezialfall eines Rechners als Prozessor zu, um zu versuchen, aus dem allgemeinen Modell Forderungen für den Rechner-Betrieb abzuleiten.

Hierbei soll zunächst noch nicht die Frage nach "hard-ware" oder "soft-ware"-Realisierung gewisser Verwaltungsaufgaben und auch nicht die Frage nach der Realisierung in einem Betriebssystem oder im Anwendungsprogramm diskutiert werden.

Die Gesamtheit der in einem System ablaufenden Prozesse können offenbar in rechnerinterne Prozesse (die den Rechner als Prozessor benötigen) und rechnerexterne Prozesse (die den Rechner nicht als Prozessor benötigen) aufgeteilt werden. Rechnerinterne Prozesse werden als Tasks bezeichnet. Zum Bereich der Rechnerverwaltung gehören alle Hilfsmittel des Systems, die entweder von den Tasks selbst oder von solchen externen Prozessen benötigt werden, die Tasks über explizite oder implizite Kopplung von einander abhängig machen. Im allgemeinen ist die Menge der zu verwaltenden Betriebsmittel größer als die Menge der von den Tasks direkt benötigten. Untersuchungen solcher Strukturen gibt es meines Wissens noch nicht; genauere Aussagen über den Umfang der Verwaltung kann ich daher nicht machen.

Verwaltungsaufgaben entstehen genau bei Beginn und Ende von Prozeßstücken, da per Definition an diesen Stellen für Prozeßabläufe relevante Änderungen von Hilfsmittelzuständen eingetreten sind. Auf den Rechner bezogen kann man wieder von internen bzw. externen Ereignissen sprechen, je nachdem, ob Beginn oder Ende eines Taskstückes bzw. eines externen Prozeßstückes vorgelegen hat.

Die folgenden Anforderungen an die Verwaltung können unterschieden werden (die Reihenfolge ist zufällig):

1. Lokale Hilfsmittelverwaltung.

   Hierunter verstehe ich die Aufgabe, eine Task genau dann fortzusetzen, wenn alle für die Fortsetzung erforderlichen Hilfsmittelzustände vorliegen.

2. Gobale Hilfsmittelverwaltung:

   Hierunter verstehe ich die Überwachung, ob die Vergabe eines Hilfsmittels an eine anfordernde Task bei dem derzeitigen Zustand der Hilfsmittel des Sytems und aufgrund der zu erwartenden Entwicklung zu einer unauflösbaren Blockadesituation führen kann und die Vergabe des Hilfsmittels deshalb aufgeschoben werden muß, bis die Blockadegefahr vorüber ist.

3. Prozessor-Verwaltung

   Hierunter verstehe ich das für das unterbrechbare Hilfsmittel Prozessor durchzuführende Retten bzw. Restaurieren des Prozessorzustandes bei Beginn bzw. am Ende einer Prozessor-Unterbrechung.

4. Start-Überwachung

   Hierunter verstehe ich die Prüfung, ob eine neu in des System aufzunehmende Task formal richtigen Gebrauch von allen Hilfsmitteln macht, also zum Beispiel nicht einen Hilfsmittelzustand verlangt, der nie eintreten kann.

5. Stop-Überwachung

   Hierunter verstehe ich die bei Beendigung einer Task erforderliche (automatische) Freigabe aller Hilfsmittel, die die Task belegt hatte, sowie die nötigenfalls gewaltsame Terminierung aller Tasks, die noch auf Hilfsmittelzustände warten, welche nur durch die beendete Task hätten herbeigeführt werden können.

6. Ablaufsteuerung

   Hierunter verstehe ich die Entscheidung zwischen alternativ möglichen Zuweisungen von Hilfsmitteln an mehrere anfordernde Tasks. Eine Zuweisung gilt als möglich, wenn die globale und lokale Hilfsmittelverwaltung die Zuweisung genehmigen würde. Über die Ablaufsteuerung kann das Verhalten der Prozesse im Hinblick auf vorgegebene Kriterien optimiert werden.

Im Prinzip könnte man sich eine solche Verwaltung realisiert denken durch ein spezielles Programm im zu verwaltenden Rechner selbst. Dieses Programm wird am Ende jedes Prozeßstückes durch "hard-ware" verwaltete Prozessor-Unterbrechung in Kontrolle gesetzt und entscheidet mittels Tabellen, die sowohl den gegenwärtigen Zustand aller zu verwaltender Hilfsmittel wie auch deren für die Fortsetzung der nächsten Taskstücke erforderlichen Zustände enthalten, welches Taskstück als nächstes den Prozessor erhalten und damit fortgesetzt werden soll.

Praktisch ist bisher eine solche Lösung nicht gelungen. Abgesehen von den formal noch nicht untersuchten Strukturproblemen der hier beschriebenen Struktur (was ein Problem der Graphentheorie - speziell ein Transportproblem - wäre), treten Probleme einerseits im Zusammen-

hang mit der Definition des Zustandes eines Hilfsmittels und andererseits der Vorausbestimmtheit der für die Fortsetzung eines Prozeßstückes erforderlichen Zustände von Hilfsmitteln auf.

Existierende Betriebssysteme helfen sich hier zum Beispiel:

1. durch Einschränkung der zwischen Prozessen erlaubten Abhängigkeiten,
2. durch Regeln für die Reihenfolge von Hilfsmittelzugriffen (welche i.A. zu nicht optimalen Betriebsmittel-Benutzungen führen),
3. durch Reduktion der Kommunikation auf bloßen Signal-Austausch.

Die unter 2., 4. und 5. aufgeführten Forderungen an eine multi-Tasking-Verwaltung bleiben dabei dann ganz oder teilweise auf der Strecke.

Im Zusammenhang mit der Frage der Realisierung der Verwaltungsaufgaben steht auch die Frage nach einer optimalen Verwaltungsstruktur. Lassen sich zum Beispiel die Prozessor-Tasks je nach ihrer Hilfsmittelbenutzung einteilen in eine Menge von Tasks, die nur eine kleine Teilmenge der vorhandenen Hilfsmittel benutzt, und in einen alle Hilfsmittel benutzenden Rest, wird sicher eine hierarchische Verwaltungsstruktur angebracht sein. Systematische Untersuchungen dieses Aspekts existieren m.W. aber nicht.

4. Multi-Tasking-Verwaltung und deren Realisierungsmöglichkeit auf kleinen Rechnern.

Im folgenden soll eine für Kleinrechner akzeptable Kompromißlösung für Multi-Tasking beschrieben und ihre mögliche Realisierung umrissen werden.

Kleinrechner werden bevorzugt zur Prozeßkontrolle und Meßwerterfassung eingesetzt. Die Aufgabe eines Kleinrechners steht entweder von Beginn an fest, oder ist zumindest immer vom gleichen Typ; das gilt auch für die Geräteumgebung des Rechners. Der Umfang der Aufgabenstellung ist überschaubar; entsprechend ist der Dokumentationsaufwand für die zur Lösung benutzten Programme erträglich. Die Zahl der für die Programmierung zuständigen Leute ist klein, der Wechsel der Programmierer selten.

Unter diesen Umständen kann auf die folgenden Anforderungen an die Verwaltung verzichtet werden:

Forderung 2. Globale Hilfsmittel-Verwaltung

Forderung 4. Start-Überwachung

Forderung 5. Stop-Überwachung.

Diese Forderungen können aufgegeben werden, da der einzelne Programmierer den Ablauf der Hilfsmittel-Anforderungen und Bereitstellungen in seinem Programm übersehen und ausreichend dokumentieren und aus der Dokumentation seiner Programmiererkollegen deren Hilfsmittelanforderungen und Bereitstellungen entnehmen kann.

Die Möglichkeit von Hilfsmittel-"dead locks" und falsche Voraussetzungen über Hilfsmittel-Bereitstellung sind damit auf die Ebene von Programmierfehlern abgeschoben. Das gilt auch für die bei Task-Ende erforderlichen "Bereinigungs"-Aufgaben.

Randbemerkung: Genau diese Forderungen sind auch bei den mir bekannten Großrechnerbetriebssystemen weitgehend aufgegeben worden, obwohl hier die bei Kleinrechnern gegebenen Voraussetzungen nicht bestehen.

Für die noch verbleibenden Verwaltungsfunktionen kann von individuellen Hilfsmittelzuständen völlig abstrahiert werden. Es genügen je zu verwaltendem Hilfsmittel und je zu verwaltender Kommunikation zwischen Prozessen über Hilfsmittelzustände die folgenden Angaben zu machen:

1. ist erwarteter Zustand eingetreten, ja oder nein?
2. wird der Zustand durch Weitermeldung zerstört, ja oder nein? (Hilfsmitteleigenschaft)
3. welchen Tasks ist Eintreten des Zustandes mitgeteilt worden, und wird von ihnen als unzerstört vorausgesetzt?
4. welche Tasks warten auf Eintreten des Zustandes und beabsichtigen diese, den Zustand zu zerstören?

Die Information über die Individualität des Hilsmittels und des mitgeteilten oder erwarteten Zustandes sind vollständig benutzerprogrammintern. Das zu verwaltende Objekt wird der Verwaltung gegenüber repräsentiert durch eine vom Benutzer einzurichtende Liste für die obigen Angaben.

Ein einziges Hilfsmittel ist dem Verwaltungsprogramm von vornherein bekannt und wird individuell verwaltet, nämlich der Prozessor selbst. Die Prozessorverwaltung umfaßt eine Liste der konkurrierenden Tasks, sowie den für die Fortsetzung einer Task erforderlichen Prozessor-Zustand.

Der Ablauf der Verwaltung geschieht über Verwaltungsprogramm-Anrufe (am einfachsten realisiert durch intern oder extern verursachte Programmunterbrechungen), bei denen als Parameter die Listen-Adressen der zu verwaltenden Objekte, sowie der Anrufgrund mitgeteilt werden. Anrufgründe sind: Warten auf Eintreten eines Hilfsmittelzustandes unter Angabe, ob Zerstörung des Zustandes beabsichtigt ist, oder Meldung des Eintretens eines Hilfsmittelzustandes.

Zum Schluß dieses Kapitels noch eine Bemerkung zur Ablaufsteuerung: Bei kleinen Rechnern genügt hierzu meist eine an die Task gekoppelte Zahl, die die Tasks nach Prioritäten ordnet. Bei alternativen Möglichkeiten der Hilfsmittelvergabe bevorzugt die Verwaltung dann die anfordernde Task mit der höchsten Priorität. Die Taskprioritäten werden vom Programmierer zugeordnet, um das von ihm gewünschte Verhalten des Systems zu erreichen.

Wesentlich schwieriger ist das Problem der Ablaufsteuerung bei großen Anlagen mit variablen Lastenverhältnissen an den System-Hilfsmitteln. Hier muß aus der gegebenen Situation eine von Fall zu Fall neue optimale Ablaufsteuerung möglich sein, um den Rechner gut zu nutzen.

5. Beispiel eines Multi-Task-Systems für einen kleinen Rechner

Als Beispiel einer Multi-Task-Verwaltung in einem Klein-Rechner-Betriebssystemmöchte ich das bei DESY für den Rechner PDP-8 der Firma Digital Equipment entwickelte System benutzen.

Die PDP-8 in der bei DESY verwendeten Ausbaustufe hat mindestens 8K Kernspeicherworte zu je 12 bit und gehört damit in die Kategorie der Kleinrechner.

Die Ideen für die Anforderungen an die Taskverwaltung wurden der Beschreibung der "IBM 360/OS Concepts and Facilities" [2] entnommen. Detailierte Berichte über das Konzept des PDP-8-Systems und die Struktur der Task-Verwaltung wurden 1966[3,4] und 1969[5,6] veröffentlicht.

Hier möchte ich die Taskverwaltungsverfahren im Lichte der vorangegangenen allgemeinen Erörterungen erläutern.

5.1 System-Teile und - Funktionen

Innerhalb des Systems werden vier wesentliche Teile unterschieden:

a) Unterbrechungs-Supervisor

b) Expreß-Aktionen

c) Task-Supervisor

d) Tasks

Die Teile a) und c) sind Standard-Bestandteile des Systems, b) und d) sind vom Benutzer zugefügte Teile und zur Erledigung problemspezifischer Aufgaben bestimmt.

Expreß-Aktionen und Tasks unterscheiden sich in ihren Privilegien und in der Geschwindigkeit, mit der sie vom System in Kontrolle gebracht werden. Expreßaktionen laufen unterbrechungsfrei und werden sehr schnell bedient, dürfen aber außer dem Prozessor selbst keine internen Hilfsmittel benutzen. Tasks benötigen mehr Verwaltungsaufwand seitens des Systems, da sie Zugriff zu allen vom System verwalteten Hilfsmitteln verlangen dürfen, erfordern dadurch aber längere Bedienungszeiten.

Die Verwaltungsaufgaben auf der Tasks-Ebene werden vom Task-Supervisor durchgeführt.

Der Task-Supervisor empfängt **Aufträge** über eine Auftragsliste. Einträge in diese Auftragsliste erfolgen durch die Expreß-Aktionen.

Die folgenden Verwaltungsfunktionen stehen zur Verfügung:

1. Task ATTACH

   Eine neue Task wird dem Supervisor zur Verwltung übergeben

2. Task RETURN

   Eine Task hat ihre Aufgabe beendet und verschwindet aus der Konkurrenz um den Prozessor und seine Hilfsmittel.

3. WAIT auf Event

   Eine Task geht in Wartestellung auf ein Signal.

4. POST Event

   Ein Signal wird erzeugt.

5. ENQUEUE für Hilfsmittel

   Eine Task verlangt alleinigen Zugriff zu einem Hilfsmittel.

6. DEQUEUE für Hilfsmittel

   Eine Task gibt ein Hilfsmittel wieder frei.

Die Funktionen-Paare WAIT/POST bzw. ENQUEUE/DEQUEUE gestatten die Realisierung der wichtigsten **expliziten** bzw. **impliziten** Abhängigkeiten zwischen Prozeßstücken. WAIT/POST gestattet die Kommunikation über das Eingetretensein gewisser Hilfsmittel-Zustände; ENQUEUE/DEQUEUE gestattet die Reservierung von Hilfsmitteln für exklusive Benutzung durch jeweils nur eine Task.

5.2. Prinzip der Task-Verwaltung

Eine Task, soweit sie dem Supervisor überhaupt bekannt ist, kann nur in genau einem von drei Zuständen sein:

1. aktiv
2. bereit zur Aktivierung
3. wartend auf Eintreten einer Bedingung
   (Eintreffen eines Signals oder Freiwerden eines Hilfsmittels).

Entsprechend existiert im Supervisor

1. ein Platz, der auf die gerade aktive Task verweist,
2. eine Liste aller Tasks, die bereit zur Aktivierung sind (Bereit-Liste),
3. für jedes Signal und jedes Hilfsmittel eine Liste der Tasks, die auf die entsprechende Bedingung warten.

Alle Listen des Supervisors sind identisch aufgebaut. Sie bestehen aus einem Listen-Kontrollblock und angeketteten Listenelementen, die die Listeneinträge repräsentieren.

Tasks werden durch einen Task-Kontrollblock repräsentiert (TCB). Da nur Tasks sich in die verschiedenen Listen eintragen können, und zwar jede Task nur in höchstens eine Liste zur gleichen Zeit, ist ein Listenelement Bestandteil des TCB.

Fig. 1 zeigt die Methode der Listen-Verkettung.

Die Verwaltungsfunktionen des Tasksupervisors führen in Abhängigkeit von der verlangten Funktion und dem Zustand der verschiedenen Listen zu neuen Listenzuständen und u.U. zur Änderung des Platzes, der auf den TCB der aktiven Task verweist (siehe Fig. 2).

Zusammenspiel der Komponenten

In Fig. 3 ist die Einbettung der Expreßaktionen und des Task-Supervisors in den Unterbrechungs-Supervisor dargestellt.

Alle Anrufe an das System werden durch eine Programmunterbrechung eingeleitet (bei externen Anrufen durch Signale von außen; bei internen Anrufen durch eine programmerzeugte Unterbrechung, genannt Supervisor-Call).

Der Anrufgrund ist zunächst in Hardware-Registern (Unterbrechungs-Flipflops) gespeichert. Das Programm rettet zuerst den Maschinenzustand und analysiert dann den Anrufgrund. In einer Liste ist der Einsprung in eine zum jeweiligen Unterbrechungsgrund gehörige Expreßroutine gespeichert.

Die Expreßroutinen können Eintragungen in die Auftragsliste des Task-Supervisors vornehmen.

Nach Ende der Expreßroutine wird geprüft, ob Task-Supervisor-Aufträge existieren. Wenn ja, wird der bei Unterbrechungsbeginn gerettete Maschinenzustand in den TCB der gerade aktiven (d.h. unterbrochenen) Task umgespeichert, und der Task-Supervisor angesprungen.

Im Task-Supervisor sind Unterbrechungen zugelassen; ein programmierter Schalter sorgt jedoch dafür, daß kein mehrmaliger Anruf des Task-Supervisors erfolgt, sondern gegebenenfalls sofort wieder in die Unterbrechungsstelle zurüchgekehrt wird.

Das Ergebnis der Task-Supervisor-Aktionen kann ein abgeänderter Zeiger auf den TCB der aktiven Task sein. Da nach Rückkehr aus dem Supervisor das Restaurieren des Maschinenzustandes aus dem TCB der aktiven Task erfolgt, kann also der Prozessor u.U. nach der Unterbrechung eine andere Task als vor der Unterbrechung fortsetzen.

.4. Erfahrungen mit diesem System

Das hier beschriebene Multi-Task-System wird seit etwa 1966 in allen bei DESY zur Experimentekontrolle benutzten PDP-8 Rechnern verwendet, und hat sich bestens bewährt. Das Programm benötigt etwa 300 Plätze.

Literatur

[1] E.W. Dijkstra, " Co-operating Sequential Processes" in "Programming Languages", edidet by F. Genuys, Academic Press 1968

[2] IBM Operating System/360, Concepts and Facilities, Form C28-6535-0

[3] F.E. Akolk, "Some Aspects of the Use of PDP-8's as On-Line-Computers in High Energy Experiments at DESY", DECUS proceedings of the Second European Seminar, Aachen 1966

[4] H.Frese, "PDP-8 Supervisor (Task Management)", ebenda

[5] F.Akolk, G.Hochweller, "Editor program SUPEDT", Interner Bericht DESY F58-69/1

[6] F.Akolk, G.Hochweller, "Systemanweisungen", Interner Bericht DESY F58-69/2

Listen-Kontrollblock (LCB)

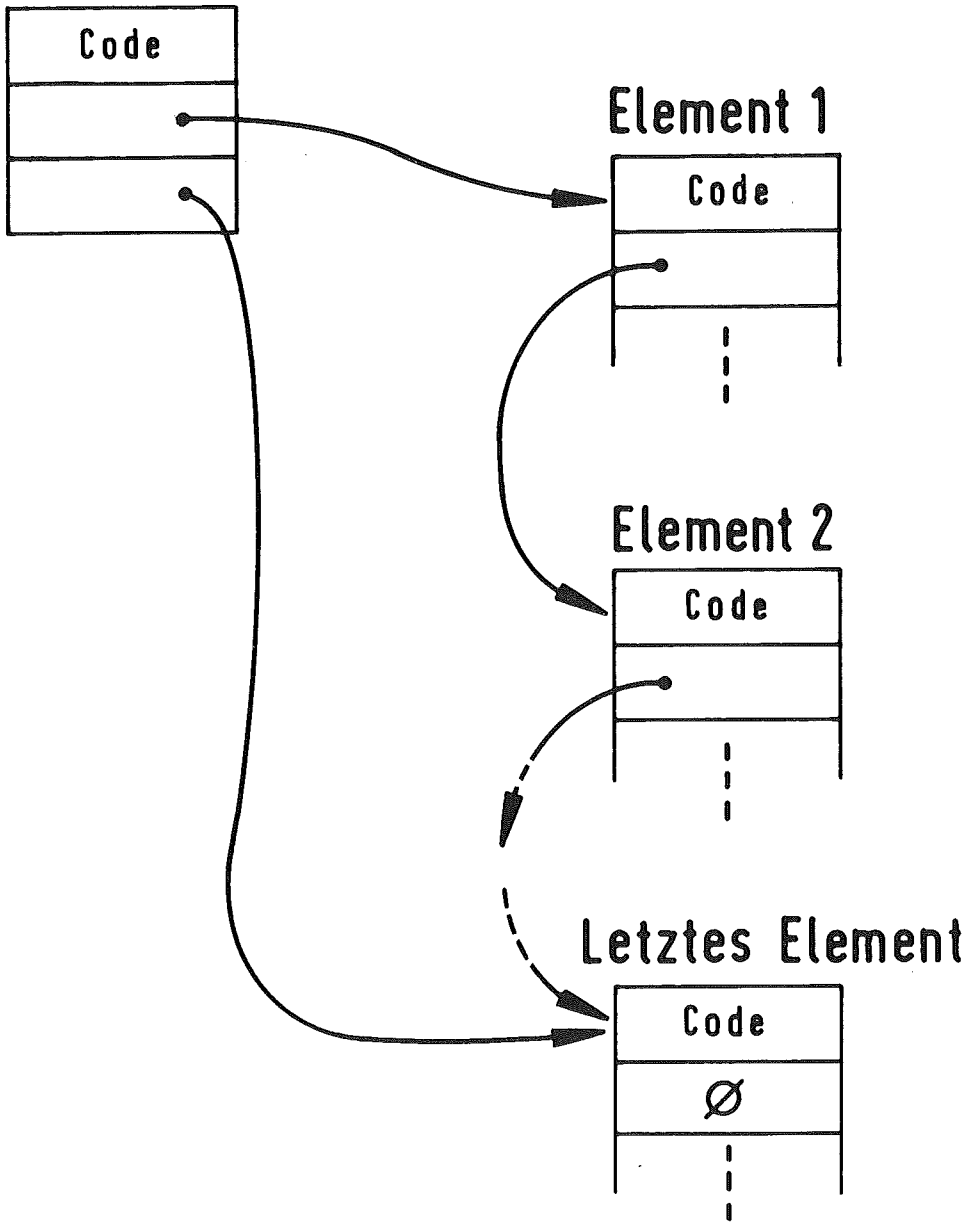

Fig.1 Standard-Format der Supervisor-Listen

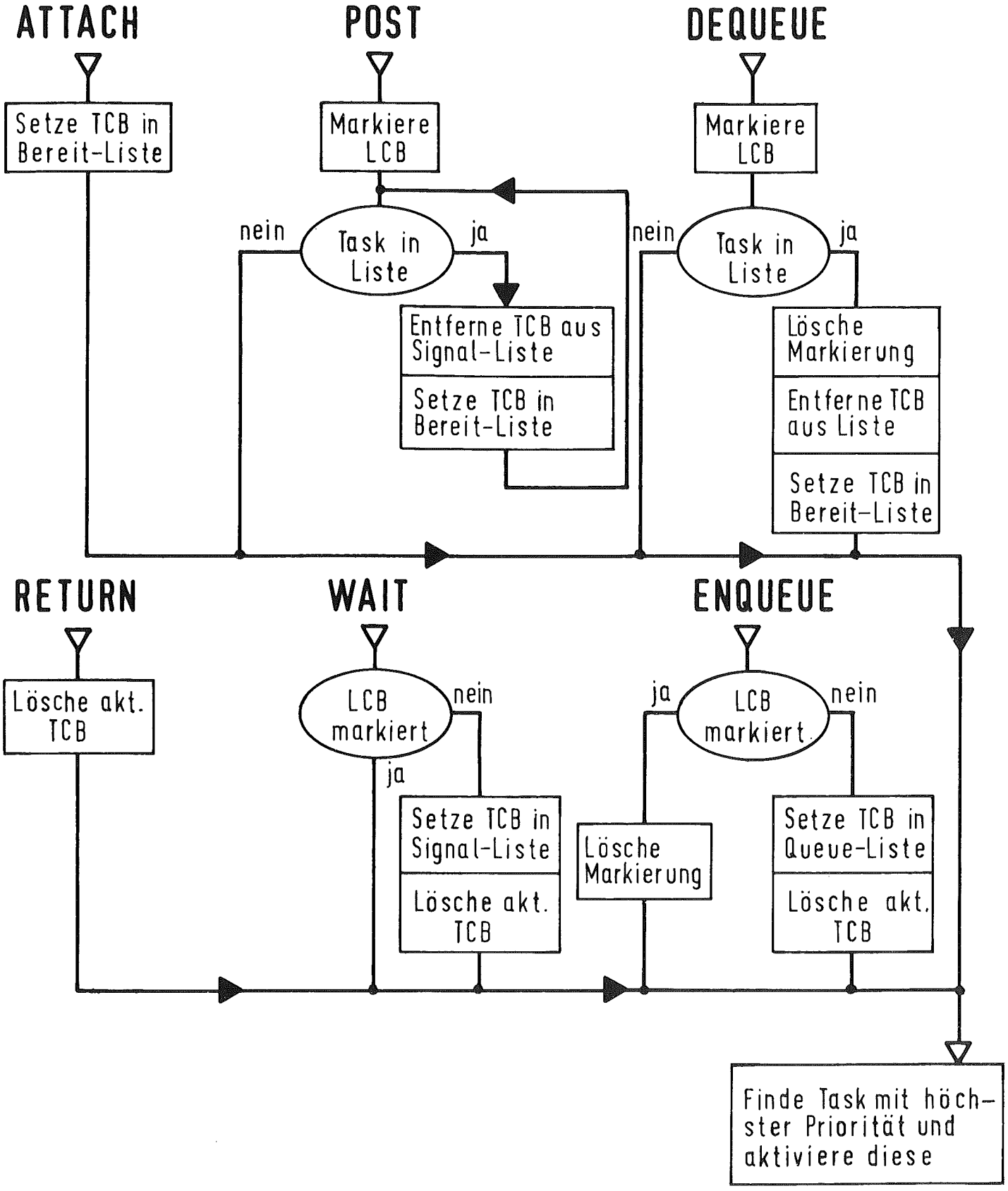

Fig. 2 Flußdiagramm der Supervisor-Funktionen

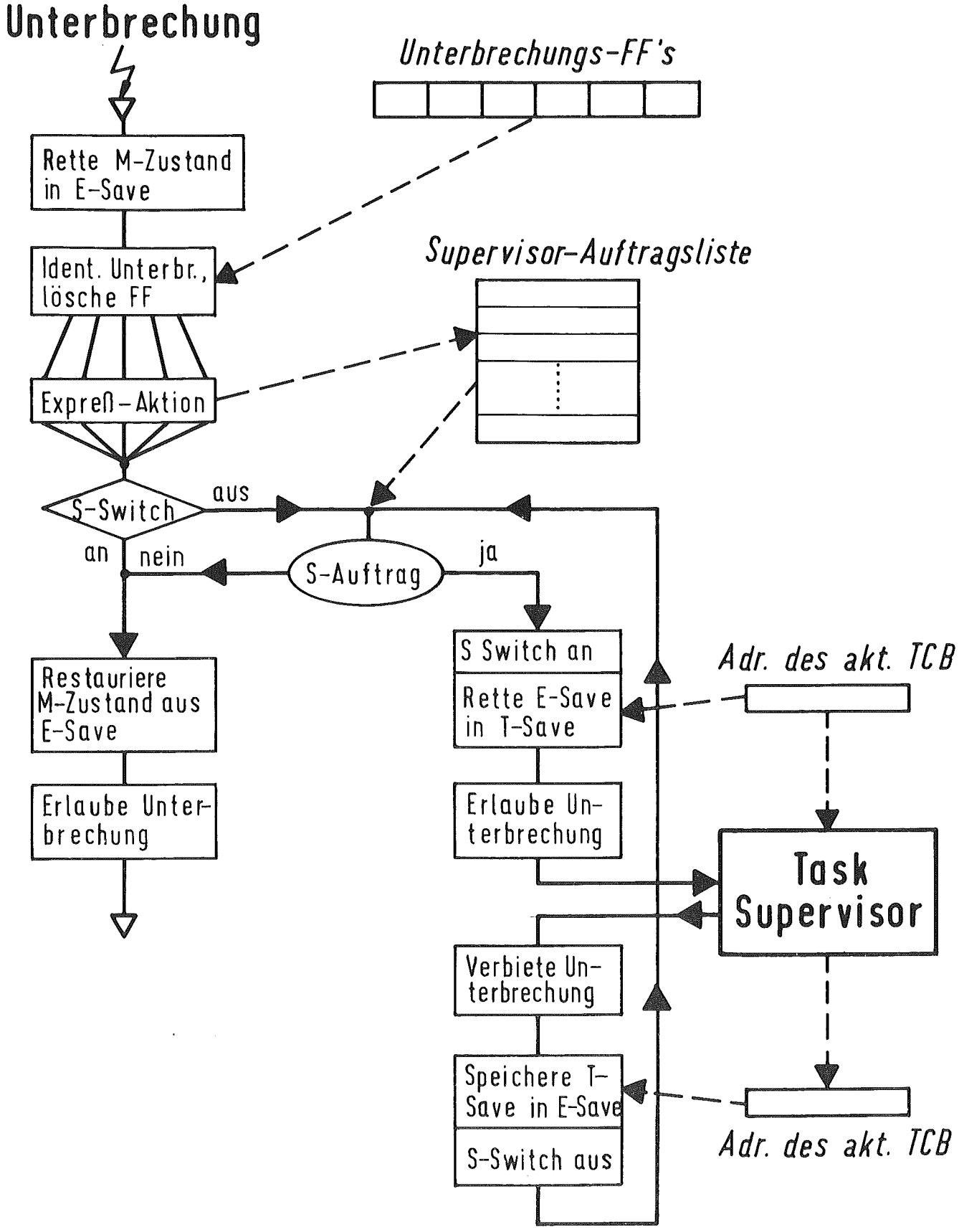

Fig.3 Zusammenhang der Systemteile