

Betriebssystemstrategien zur  
Bewältigung von Zeitproblemen  
in der Prozeßautomatisierung

von der Universität Stuttgart  
zur Erlangung der Würde eines  
Doktor-Ingenieurs (Dr.-Ing.)  
genehmigte Abhandlung

vorgelegt von  
ROLAND RÖSSLER  
geboren in Linz / Österreich

Hauptberichter: Prof. Dr.-Ing. R. Lauber  
Mitberichter: Prof. Dr.-Ing. A. Lotze

Tag der Einreichung: 24.4.1978  
Tag der mündlichen Prüfung: 30.4.1979



### Kurzfassung

Die Zeitprobleme, die bei der Anwendung von Rechnern für die Prozeß-automatisierung auftreten, werden klassifiziert und an Hand von praktischen Beispielen diskutiert. Bereits eingeführte Strategien zur Verwaltung von parallelen Aktivitäten werden auf ihre Eignung zur Bewältigung dieser Anforderungen diskutiert. Es werden ergänzende Maßnahmen vorgeschlagen, die diese Fähigkeiten verbessern. Über die Implementation eines Modellbetriebssystems werden einige der diskutierten Strategien realisiert und über praktische Versuche verglichen.

### Vorwort

Ich danke Herrn Prof.Dr.R.Lauber herzlich für seine Unterstützung während der Entstehung dieser Arbeit. Den Herren Angerer und Hähle von der Firma SIEMENS AG - Erlangen danke ich für wertvolle Hinweise auf zeitkritische Anwendungsfälle in der Praxis. Schließlich danke ich Herrn Professor Dr. A. Lotze für die freundliche Übernahme des Mitberichtes.

## Inhaltsverzeichnis

Literaturverzeichnis	6
Verzeichnis der Abkürzungen	9
1. EINLEITUNG	11
1.1. Hintergrund	11
1.2. Thematik	12
1.3. Begriffsbestimmung	13
2. ZEITPROBLEME BEIM EINSATZ VON RECHNERN FÜR DIE PROZESSAUTOMATISIERUNG	15
2.1. Vorbemerkungen	15
2.2. Die Reaktionszeit als Bewertungsgrundlage	18
2.3. Leistungskriterien für Prozeßrechner	19
2.4. Klassifikation von zeitlichen Anforderungen	20
2.5. Praktische Beispiele	23
2.5.1. Walzstraßenautomatisierung	23
2.5.2. Direkte Digitale Computerregelung (DDC)	25
2.5.3. "Zielbremssteuerung"	27
2.5.4. Dynamische Prüfung	28
2.5.5. Meßwertüberwachung auf gleitende Grenzen	30
2.5.6. Prüfstände	31
2.5.7. Ablaufberg	32
2.5.8. Anfahrvorgänge	32
2.6. Lösungswege	33
3. TASKVERWALTUNGSSTRATEGIEN FÜR PROZESSRECHNER	38
3.1. Beschreibung der Strategien	38
3.1.1. Feste Prioritäten ("FP")	38
3.1.2. "Dynamische Prioritäten"	41
3.1.3. First-in-first-out ("FIFO")	42
3.1.4. Strategie "nach der kürzesten Restzeit" ("KR")	44
3.1.5. Strategie "nach dem kleinsten Spielraum" ("KS")	49
3.1.6. Der Begriff der "Restzeitreduktion"	50
3.1.7. Berücksichtigung "langsamer" Ein/Ausgabe	52
3.1.8. Berücksichtigung von "kritischen Abschnitten"	54

3.2. Implementations- und Anwendungsaspekte	55
3.2.1. Feste Prioritäten ("FP")	55
3.2.2. Dynamische Prioritäten	58
3.2.3. First-in-first-out ("FIFO")	59
3.2.4. Strategie nach der kürzesten Restzeit ("KR")	59
3.2.5. Strategie nach dem kürzesten Spielraum ("KS")	62
3.2.6. "Restzeitreduktion"	62
3.2.7. "Kritische Prioritäten"	63
4. IMPLEMENTATION VON TASKVERWALTUNGSSTRATEGIEN	64
4.1. Allgemeines	64
4.1.1. Fähigkeiten des Modellbetriebssystems	64
4.1.2. Implementationstechnik	70
4.2. Grundsätzlicher Aufbau	70
4.3. Realisierung der Taskverwaltungsstrategien	73
4.3.1. Strategie "First-in-first-out" ("FIFO")	73
4.3.2. Feste Prioritäten ("FP")	74
4.3.3. Strategie nach der kürzesten Restzeit ("KR")	74
4.3.4. Zusatzbedingung "Restzeitreduktion"	74
4.3.5. "Kritische Prioritäten"	75
5. PRAKTISCHE ERPROBUNG VON TASKVERWALTUNGSSTRATEGIEN	81
5.1. Funktionsweise des Prozeßrechnertestgerätes	81
5.2. Vorgangsweise	83
5.3. Versuchsreihen	88
5.3.1. Auswahl der Parameter und Ergebnisse	89
5.3.2. Erste Versuchsreihe	90
5.3.3. Dritte Versuchsreihe	95
5.4. Kritik der Versuchsergebnisse	97
Zusammenfassung	98
ANHANG I	
Ableitungen zu den Kapiteln 3.1.6. und 3.1.7.	99

ANHANG II	
Beschreibung der Bausteine des Modellbetriebssystems	106
II.1. Verwaltungsdaten	106
II.1.1. Taskverwaltungsblock	109
II.1.2. Bolt-Verwaltungsblock	110
II.1.3. Semaphor-Verwaltungsblock	111
II.1.4. Geräteverwaltungsblock	111
II.1.5. Auftragsparameterblock	112
II.1.6. Interruptliste	114
II.1.7. Zeitliste	115
II.2. Verwaltungsbausteine	115
II.2.1. Prozessorverwaltung	115
II.2.2. Unterbrechungsverwaltung	116
II.2.2.1. Routinen für Zeitgeber und Alarm-Unterbrechung	116
II.2.2.2. Fertigmeldungen von Geräten	117
II.2.3. Ein/Ausgabeverwaltung	119
II.2.4. Einplanungsverwaltung	121
II.2.5. Taskverwaltung	123
II.2.5.1. Der Modul ACTIVATE	123
II.2.5.2. Der Modul TERMINATE	123
II.2.5.3. Der Modul CONTINUE	127
II.2.5.4. Der Modul RESUME	128
II.2.5.5. Der Modul PREVENT	129
II.2.5.6. Die Moduln REQUEST und RELEASE	130
II.2.5.7. Die Moduln RESERVE und FREE	131
II.2.6. Auftragsverwaltung	133

Literaturverzeichnis

- BAE76 Baeker, K.: Entwurf und Bau eines Prozeßrechnertestgerätes. Diplomarbeit, Univ. Stuttgart, 1976
- CAM74 Campbell, R.H., Lauer, P.E.: Formal Semantics of a Class of High-Level Primitives for Coordinating Concurrent Processes. Acta Informatica 5 (1975) 297-332
- DIJ72 Dijkstra, E.W.: Hierarchical Ordering of Sequential Processes. Operating System Techniques, Ac. Press London und New York (1972) S. 72-93
- EIC75 Eichenauer, B.F.: Dynamische Prioritätsvergabe an Tasks in Prozeßrechnersystemen. Diss. Univ. Stuttgart 1975.
- GHA77 Ghassemi, A.: Untersuchung der Eignung der Prozeßprogrammiersprache PEARL zur Automatisierung von Folgeprozessen. Eingereicht als Dissertation, Univ. Stuttgart (1977)
- HEL74 Hellermann, L.: The power and efficiency of a computer system. Lecture Notes in Computer Science 8 (1974) 190-205, Springer Verlag, Berlin-Heidelberg-New York.
- HEN75 Henn, R.: Deterministische Modelle für die Prozessorzuteilung in einer harten Realzeit-Umgebung. Diss. Univ. München (1975)
- HER75 Herzog, U.: Optimal Scheduling Strategies for Real-Time Computers. IBM Journal of research and development 19(1975) 494-504.
- HEW74 Hewbold, P. et al.: HAL/S Language Specification. Intermetrics Inc., Report N<sup>o</sup> IR-61-5, (Nov. 1974)
- HOF74 Hofmann, F.: Vorlesung zu Betriebsprogrammierung I, Univ. Erlangen (1974)
- KRA75 Krayl, H., Neuhold, E., Unger, C.: Grundlagen der Betriebssysteme. De Gruyter, Berlin (1975)
- KUE72 Kümmerle, K.: Charakteristische Größen zur Beschreibung der Leistungsfähigkeit und Effektivität von EDV-Anlagen. Elektronische Rechenanlagen 14 (1972) H.1, 12-18

- KUE76 Küttner, T.: Programmierung, Test und Inbetriebnahme des Prozeßrechnertestgerätes. Diplomarbeit, Univ. Stuttgart, 1976
- LAU74 Lauber, R.: Leistungskriterien von Prozeßrechensystemen. Lecture Notes in Computer Science 12 (Fachtagung Prozeßrechner 1974), 586-601, Springer Verlag Berlin-Heidelberg-New York.
- LAU76 Lauber, R.: Prozeßautomatisierung I. Hochschultext, Springer Verlag, Berlin-Heidelberg-New York (1976)
- LAU77 Lauber, R.: Vorlesung zu Prozeßautomatisierung II, Univ. Stuttgart (1977)
- LIU73 Liu, C.L., Layland, J.W.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. JACM, Vol.20, No.1 (Jan.1973) 46-61.
- NEH75 Nehmer, J.: Dispatcher primitives for the construction of operating system kernels. Acta Informatica 5 (1975) 237-255
- NEH76 Nehmer, J., Eggenberger, O.: Hardwarenahe Elementarfunktionen der Ablaufsteuerung für Prozeßrechnerbetriebssysteme. PDV-Bericht KFK-PDV 49 (1976), Gesellschaft für Kernforschung mbH Karlsruhe
- PDV77 Full PEARL Language Description. PDV-Bericht 130 (1977), Gesellschaft für Kernforschung mbH, Karlsruhe
- ROE77 Rössler, R.: Beschreibung der Programmiersprache "SPASS". Interner Bericht des 3. Physikalischen Instituts der Universität Erlangen (1977)
- SER72 Serlin, O.: Scheduling of time critical processes. Spring Joint Computer Conference (1972) 925-932
- SIE69 SIEMENS SYSTEM 300 Organisationsprogramm. Beschreibung (März 1969) Bestell-Nr. D13/3006.
- VDI76 VDI/VDE-GMR-Richtlinie Nr.3552 "Leistungskriterien von Prozeßrechensystemen". VDI-Verlag, Düsseldorf (1976)
- VDI76A VDI/VDE-GMR-Richtlinie Nr.3554 "Funktionelle Beschreibung von Prozeßrechner-Betriebssystemen". VDI-Verlag, Düsseldorf (1976)

WAI70 Waite, W.M.: The mobile programming system STAGE2. CACM Vol.13  
(1970) 415-421

WET74 Wettstein, H.: Prozeßumschaltung in Betriebssystemen.  
Computing 12 (1974) 363-382



# Verzeichnis der Abkürzungen

- $B_i$ .....Boltvariable  $i$  (Synchronisierungsvariable für kritische Abschnitte)
- $\mathcal{L}(t)$ ....Bereitmenge - Menge der zum Zeitpunkt  $t$  um den Prozessor konkurrierenden Tasks (  $\mathcal{L}(t) \subseteq \mathcal{M}(t)$  )
- E/A.....Ein/Ausgabe
- $\xi$ .....Zeittoleranz (zulässige Abweichung von einem Sollzeitpunkt)
- FIFO....Strategie "First-in-first-out"
- FP.....Strategie nach "festen Prioritäten"
- KR.....Strategie nach der "kürzesten Restzeit"
- KS.....Strategie nach dem "kürzesten Spielraum"
- $l_i(t)$ ...Laufzeit der Task  $T_i$  - Zeitspanne, die die Task zum Zeitpunkt  $t$  für ihre vollständige Ausführung noch benötigen würde, wenn ihr alle Betriebsmittel ausschließlich zur Verfügung stünden
- $\mathcal{M}(t)$ ...Taskmenge - Menge der zum Zeitpunkt  $t$  betrachteten Tasks
- NACHF...Zeiger auf nächstes Element einer Warteschlange
- $P_i$ .....Priorität der Task  $T_i$
- $\overline{P_i}$ .....Prioritätszahl der Task  $T_i$  (kleineres  $\overline{P_i}$  entspricht größerer Dringlichkeit)
- $\overline{P_{k_i}}$ ....."kritische" Prioritätszahl - Prioritätszahl, die der Task  $T_i$  innerhalb eines kritischen Abschnitts zugeteilt ist
- $\overline{P_{o_i}}$ ....."Original-" Prioritätszahl - Prioritätszahl, die der Task  $T_i$  außerhalb von kritischen Abschnitten zugeteilt ist
- R.....Regler
- $r_i(t)$ ...Restzeit der Task  $T_i$  - die zum Zeitpunkt  $t$  noch verbleibende Zeitspanne bis zur Zeitschranke  $t_{zi}$  (  $r_i(t) = t_{zi} - t$  )
- $s_i(t)$ ...Spielraum der Task  $T_i$  - Zeitspanne, um die die Task maximal verzögert werden darf, um  $t_{zi}$  nicht zu verletzen (  $s_i(t) = r_i(t) - l_i(t)$  )

$T_i$ .....Task i

$T_R$ .....Ausführungszeit eines Regelalgorithmus

$T_t$ .....Totzeit

TKB.....Taskkontrollblock, Taskverwaltungsblock

$t_A$ .....Abtastzeitpunkt

$t_e$ .....Zeitpunkt des Auftretens eines (stochastischen) Ereignisses

$t_{Si}$ .....Startzeitpunkt der Task  $T_i$  (Zustandswechsel ruhend  $\rightarrow$  bereit)

$t_{St}$ .....Stellzeitpunkt

$t_v$ .....Zeitpunkt der Vereinbarung einer Zeitbedingung

$t_{Zi}$ .....Zeitschranke der Task  $T_i$  - Zeitpunkt, zu dem sie spätestens  
beendet sein sollte

VORG....Zeiger auf vorheriges Element einer Warteschlange

WSANF...Zeiger auf erstes Element einer Warteschlange

WSEND...Zeiger auf letztes Element einer Warteschlange

w.....Führungsgröße

x.....Regelgröße, Prozeßgröße

y.....Stellgröße

z.....Störungsgröße

#### Abkürzungen im Anhang II:

BWS.....Bolt-Warteschlange

GWS.....Gerätewarteschlange

PWS.....Prozessorwarteschlange

SWS.....Semaphor-Warteschlange

TBK.....Task-Bolt-Kette (Kette der von einer Task belegten Bolts)

## 1. EINLEITUNG

### 1.1. Hintergrund

Für die Überwachung, Steuerung und Regelung von technischen Prozessen werden in immer größerem Maße Rechenanlagen eingesetzt. In den meisten Anwendungen spielen neben anderen Faktoren Zeitbedingungen eine bedeutende Rolle, was auch durch den Begriff "Realzeitbetrieb" zum Ausdruck gebracht wird, der für diesen Problemkreis vielfach verwendet wird. Diese Zeitbedingungen drücken sich darin aus, daß die Kommunikation zwischen dem Prozeßrechner und dem technischen Prozeß zeitlichen Anforderungen genügen muß, die durch die technologischen Gegebenheiten bestimmt sind. Beispielsweise sollen zusammengehörige Meßwerte vom Rechner oft möglichst gleichzeitig abgeholt werden, oder Reaktionen auf Alarmsituationen sollen rechtzeitig erfolgen.

In vielen Fällen ist das Zeitverhalten des technischen Prozesses ausreichend träge, sodaß die Erfüllung dieser Anforderungen für den betreffenden Rechner kein großes Problem darstellt. Es gibt aber genügend Beispiele, bei denen trotz der heute erzielbaren hohen Verarbeitungsgeschwindigkeiten Konfliktsituationen entstehen. Dies gilt vor allem dann, wenn ein Prozeßrechner nicht nur eine, sondern eine Vielzahl von Aufgaben simultan zu bewältigen hat, bei denen strengen zeitlichen Anforderungen Rechnung zu tragen ist.

Vielfach werden heute noch ausschließlich Hardware-Eigenschaften einer Datenverarbeitungsanlage als maßgebliche Kriterien für ihre Fähigkeit angesehen, mit solchen Problemen fertig zu werden. Für den erwähnten Fall der "gleichzeitigen" Behandlung von mehreren Teilaufgaben innerhalb eines technischen Prozesses, die in ihren zeitlichen Abläufen mehr oder weniger voneinander abhängig sein können, müssen jedoch auch Softwarekomponenten in die Betrachtung mit einbezogen werden. Es handelt sich dabei um Programmbausteine, denen die Koordinierung dieser Abläufe obliegt. Zur Bearbeitung der erwähnten Teilaufgaben werden sogenannte "Betriebsmittel" benötigt. Dabei handelt es sich um all die Elemente, die den erforderlichen Ablauf im Rechner ermöglichen - wie Rechenzeit, Speicherplatz, Ein/Ausgabe-Geräte etc. Die genannten koordinierenden Programmbausteine müssen unter anderem für die zeitgerechte Bereitstellung dieser Betriebsmittel für die einzelnen Teilaufgaben sorgen. Dabei muß der Umstand

berücksichtigt werden, daß die meisten Betriebsmittel üblicherweise nur in beschränkter Anzahl zur Verfügung stehen, daß sich also die Teilaufgaben in Konkurrenzsituationen befinden können. Es müssen dann Aktionen, die eigentlich gleichzeitig erfolgen sollten, hintereinander abgewickelt werden.

Um dabei den vorliegenden Zeitbedingungen Rechnung tragen zu können, werden Strategien eingesetzt, durch die die Bearbeitungsreihenfolge in derartigen Konfliktsituationen festgelegt wird.

Im sogenannten "Stapelbetrieb" bei Großrechnern treten ähnliche Probleme auf, die bereits in großem Umfange theoretisch durchleuchtet sind. Eine große Zahl von derartigen Strategien kann damit nach verschiedenen Kriterien beurteilt werden. Ein solches Kriterium ist zum Beispiel die "Minimierung der mittleren Verweilzeit" - das ist jene Zeitspanne, die zwischen Abgabe und Fertigstellung eines Auftrages verstreicht.

Nun hat jedoch ein Teil der im Stapelbetrieb auftretenden zeitlichen Anforderungen keine sinnvolle Entsprechung bei den Problemen der Prozeßautomatisierung - und umgekehrt. Als Folge davon sind Strategien, die die Vergabe von Betriebsmitteln regeln, nur mit Einschränkungen in beiden Anwendungsgebieten einsetzbar.

In einigen Arbeiten findet man daher auch Untersuchungen, denen die speziellen Zeitbedingungen bei Realzeitanwendungen zugrunde liegen. Da die vollständige Durchleuchtung der in der Praxis auftretenden Probleme auf theoretischem Wege mit erheblichen Schwierigkeiten verbunden ist, beschränken sich die Autoren meist auf idealisierte Verhältnisse. Eine etwas unbefriedigende Situation besteht auch deshalb, weil den diversen theoretischen Betrachtungen oft keine aktuellen Implementationen entsprechen, die für den Einsatz in der Prozeßdatenverarbeitung modellhaften Charakter haben könnten.

## 1.2. Thematik

Als Grundlage für weitere Betrachtungen werden zuallererst die zeitlichen Anforderungen, die bei der Prozeßautomatisierung in großer Vielfalt auftreten, klassifiziert. An praktischen Beispielen werden die charakteristischen Merkmale dieser Klassen erläutert.

Darauf aufbauend werden Strategien zur Vergabe von Betriebsmitteln auf ihre Eignung zur Lösung dieser Zeitprobleme untersucht. Dabei werden sowohl Strategien behandelt, die bereits in aktuellen Implementationen realisiert sind, als auch solche, die zwar bekannt sind, aber sich bisher noch kaum im Einsatz befinden. Darüberhinaus werden Maßnahmen vorgeschlagen, die diese Mechanismen ergänzen, und die bei Berücksichtigung von erschwerenden Randbedingungen eine Verbesserung des Zeitverhaltens der Programme nach sich ziehen.

Um die Kluft zwischen theoretischer Betrachtung und aktueller Anwendung teilweise zu überbrücken, wird ein Teil der untersuchten Strategien über eine Modellimplementation realisiert.

An Hand von praktischen Beispielen werden Vergleiche vorgenommen, die die gewonnenen Erkenntnisse stützen.

### 1.3. Begriffsbestimmung

In der Folge werden einige technische Ausdrücke bestimmt, die in dieser Arbeit häufig verwendet werden:

#### BETRIEBSSYSTEM

Unter einem Betriebssystem versteht man "die Programme eines digitalen Rechensystems, die zusammen mit den Eigenschaften der Rechenanlage die Grundlage der möglichen Betriebsarten des digitalen Rechensystems bilden und insbesondere die Abwicklung von Programmen steuern und überwachen" (DIN 44300, Nr. 59).

#### TASK (Rechenprozeß)

Die Aufgaben, die eine Rechenanlage bei der Steuerung bzw. Regelung eines technischen Prozesses zu bewältigen hat, können in einzelne Teilaufgaben zerlegt werden, die dadurch charakterisiert sind, daß auf Grund eines Ereignisses ein Bearbeitungsvorgang im Rechner angestoßen wird, der schließlich ein Ergebnis in seine "Umgebung" abliefert. Unter "Umgebung" kann der technische Prozeß verstanden werden - dann handelt es sich bei den Ergebnissen um die Ausgabe irgendwelcher Daten vom Rechner an die Peripherie - es kann aber auch eine rechnerinterne Umgebung betroffen sein (z.B. Datensätze).

Die Abwicklung einer solchen Teilaufgabe, oder einer streng sequentiellen Folge von zusammengehörigen Teilaufgaben durch das Betriebssystem, wird als "Task" bezeichnet. Diesem Begriff wird hier im Gegensatz zu "Rechenprozeß" zwecks besserer Unterscheidung zum "technischen Prozeß" der Vorzug gegeben.

#### TASKVERWALTUNG

Ein Prozeßrechner-Betriebssystem hat unter anderem die Aufgabe, in einer Konkurrenzsituation um ein Betriebsmittel die Reihenfolge der Vergabe desselben an die konkurrierenden Task festzulegen. Dafür ist es im allgemeinen erforderlich, für jedes Betriebsmittel eine "Warteschlange" einzurichten, in die die Tasks nach bestimmten Kriterien eingereiht werden. Diese Betriebsmittelzuteilungen und die Verwaltung der zugehörigen Warteschlangen sind Aufgabe der sogenannten "Taskverwaltung", des zentralen Bausteines eines Realzeitbetriebssystems.

#### (TASKVERWALTUNGS-) STRATEGIEN

Eine "Taskverwaltungsstrategie" stellt eine Vorschrift dar, nach der die Taskverwaltung die verfügbaren Betriebsmittel den konkurrierenden Tasks zuteilt. Der Zweck dieser Vorschrift ist die Erfüllung irgendwelcher Bedingungen - üblicherweise Zeitbedingungen - die je nach Anwendungsfall stark unterschiedlichen Charakter haben können.

## 2. ZEITPROBLEME BEIM EINSATZ VON RECHNERN FÜR DIE PROZESSAUTOMATISIERUNG

### 2.1. Vorbemerkungen

Als notwendige Voraussetzung für eine vergleichende Bewertung von Taskverwaltungsstrategien werden in den folgenden Abschnitten die Anforderungen untersucht, die in der Praxis an Prozeßrechnerbetriebssysteme in Bezug auf die Erfüllung von Zeitbedingungen gestellt werden.

Zum Verständnis der weiteren Ausführungen sei auf die in DIN 66216 definierten charakteristischen Zeitintervalle bei einer Programmunterbrechung verwiesen. Bei der in Bild 2.1 gewählten Darstellung wird der Einfachheit halber davon ausgegangen, daß die Rechenanlage zum Zeitpunkt des Auftretens des Unterbrechungssignals nicht mit der Abwicklung eines Programms so hoher (Hardware-) Priorität beschäftigt ist, daß die Programmunterbrechung zurückgestellt wird.

Die "Wartezeit", die die Summe von "Durchlaßzeit" und "Latenzzeit" darstellt, ist nahezu ausschließlich durch die Hardware der Rechenanlage bestimmt und kann demnach durch Software-Maßnahmen praktisch nicht beeinflußt werden. Ähnlich verhält es sich zum Teil mit der "Erkennungszeit": Bei vielen modernen Maschinen wird ein Unterbrechungssignal automatisch durch die Hardware identifiziert, wobei der Start des entsprechenden Antwortprogrammes über einen sog.

"Interruptvektor" vorgenommen wird. Diese Interruptvektoren enthalten unter anderem die Ansprungsadressen unter denen die betreffenden Antwortroutinen zu finden sind. Dadurch wird die Erkennungszeit praktisch auf 0 reduziert. Es muß jedoch gesagt werden, daß in den meisten Fällen dieser Mechanismus - verbunden mit einer Kellierung von Statusinformation in einem "Stack" - es nicht erlaubt, die Unterbrechungsantwortprogramme unmittelbar durch den normalen Taskverwaltungsmechanismus eines Betriebssystems mit Multiprogrammierungsfähigkeiten zu steuern. Insbesondere kann dann ein Großteil der sonst in Anwendertasks üblichen Betriebssystemaufrufe innerhalb dieser Antwortprogramme nicht abgegeben werden. Soll trotzdem aufgrund eines Unterbrechungssignals ein "normales" Automatisierungsprogramm angestoßen werden, so müssen die entsprechenden

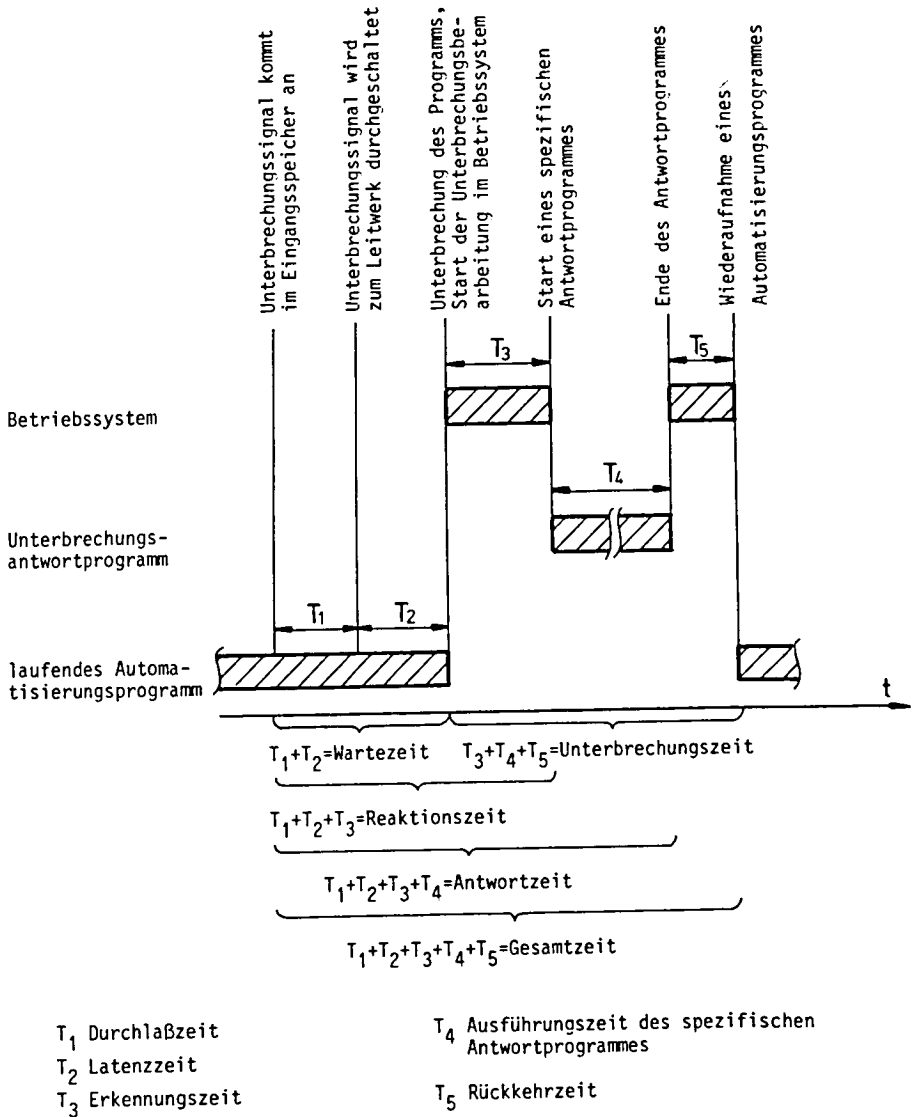


Bild 2.1: Definition charakteristischer Zeitintervalle bei der Programmunterbrechung gemäß DIN 66216



Verwaltungsprozeduren zum Start von Tasks in den Unterbrechungsantwort Routinen durchlaufen werden. Da dies wegen der gemeinsam benutzten Datenbasis (Warteschlangen, Taskverwaltungsblöcke, etc.) zum Teil nur unter gegenseitigem Ausschluß möglich ist, muß für die Dauer einer solchen Bearbeitung die Behandlung anderer Unterbrechungen (die ebenfalls Betriebssystemfunktionen ausführen sollen) unterbunden werden. Damit geht der Vorteil der Priorisierung durch die Hardware wieder verloren, und die Erkennungszeit unterscheidet sich dann nicht mehr maßgeblich von der, die sich bei älteren Verfahren (z.B. Identifikation eines Unterbrechungssignals durch explizites Abfragen von Ankunftsregistern) einstellt.

Neben den bisher genannten charakteristischen Zeitintervallen werden in den weiteren Abschnitten noch die folgenden zeitabhängigen Größen verwendet:

- . Der Startzeitpunkt  $t_{Si}$  einer Task  $T_i$  ist der Zeitpunkt, zu dem sie (erstmalig) durch das Betriebssystem in die Warteschlange der auf die Zuteilung des Prozessors wartenden Tasks eingereiht wird. Hat sie unter diesen die höchste Priorität, so beginnt damit ihre Ausführung.
- . Die Zeitschranke  $t_{Zi}$  einer Task  $T_i$  ist der Zeitpunkt, zu dem ihre Ausführung spätestens abgeschlossen sein muß.
- . Die Restzeit  $r_i(t)$  einer Task  $T_i$  ist das Zeitintervall, das ihr zum Zeitpunkt  $t$  zu ihrer Ausführung noch zur Verfügung steht.  
(  $r_i(t) = t_{Zi} - t$  )
- . Die Laufzeit  $l_i(t)$  einer Task  $T_i$  ist die Zeitspanne, die sie zum Zeitpunkt  $t$  für ihre vollständige Ausführung noch benötigen würde, wenn ihr alle Betriebsmittel ausschließlich zur Verfügung stünden.
- . Der Spielraum  $s_i(t)$  einer Task  $T_i$  ist die Differenz zwischen Restzeit und Laufzeit (  $s_i(t) = r_i(t) - l_i(t)$  ), und stellt jene Zeitspanne dar, um die die Task  $T_i$  maximal verzögert werden darf, um ihre Zeitschranke  $t_{Zi}$  nicht zu verletzen.

## 2.2. Die Reaktionszeit als Bewertungsgrundlage

Bis heute ist es in der Praxis noch vielfach üblich, zur Beurteilung der Leistungsfähigkeit eines Prozeßrechners in erster Linie die erzielbare minimale "Reaktionszeit" heranzuziehen. Dies ist in Bezug auf solche Anwendungsfälle durchaus sinnvoll, bei denen die auftretenden "Ausführungszeiten" relativ klein sind, und bei denen es fast nur darauf ankommt, die resultierende Gesamtzeit der Unterbrechungsbearbeitung möglichst klein zu halten. Sie bestimmt ja die maximale Auflösung stochastisch anfallender Ereignisse, die sich über das gleiche Unterbrechungssignal melden. Als Beispiel seien hier kernphysikalische Anwendungen erwähnt:

Bei der Spektroskopie von geladenen Teilchen, die bei Kernreaktionen entstehen, treffen in zufälliger Folge Ereignisse ein, die das Auftreffen eines Teilchens auf einen Detektor kennzeichnen und einen zur Energie proportionalen Meßwert liefern. Dieser muß vom Unterbrechungsantwortprogramm erfaßt werden, um damit das entsprechende Ereignis in ein Spektrum einordnen zu können. Wie bereits erwähnt wurde, ist bei gegebenem Rechner die Durchlaßzeit und die Latenzzeit praktisch nicht beeinflusbar. Eine mögliche Verkürzung der Gesamtzeit kann demnach softwaremäßig nur über die Erkennungszeit bzw. Rückkehrzeit im Betriebssystem und über die Ausführungszeit des Antwortprogrammes erzielt werden. Bei extrem häufig anfallenden Ereignissen empfiehlt es sich daher, das Antwortprogramm in einer (gegenüber Anwendertasks) privilegierten Ebene ablaufen zu lassen. Man spart dadurch Verwaltungszeiten, was allerdings den oben erwähnten Nachteil zur Folge hat, daß Betriebssystemaufrufe (die die Taskverwaltung betreffen) vom Antwortprogramm nicht - oder nur in eingeschränktem Umfange - abgegeben werden können, und daß in vielen Fällen durch einen unterschiedlichen Befehlsvorrat in den verschiedenen Ebenen eine einheitliche Programmierung von Automatisierungsaufgaben nicht möglich ist. Die Ausführungszeit kann unter Umständen dadurch reduziert werden, daß man sich im eigentlichen Antwortprogramm auf die Identifizierung des betreffenden Ereignisses (Messung der Teilchenenergie) und einen entsprechenden Eintrag in einen ausreichend großen Puffer beschränkt. Die eigentliche Auswertung der Daten kann dann

anschließend in einem "normalen" Programm erfolgen, das unter der Kontrolle der Taskverwaltung läuft.

Bei einer solchen Lösung spielt die für Betriebsmittelzuteilungen gewählte Strategie für die mögliche Ereignisauflösung so gut wie keine Rolle. Dies gilt natürlich insbesondere dann, wenn es sich um nur einen (solcherart "kritischen") Unterbrechungseingang handelt - wenn also Aspekte der Parallelverarbeitung von untergeordneter Bedeutung sind.

### 2.3. Leistungskriterien für Prozeßrechner

Mit dem rapiden Anwachsen der Zahl der durch Prozeßrechner zu bewältigenden Automatisierungsaufgaben und der damit verbundenen Vielfältigkeit der gestellten Anforderungen ergab sich die Notwendigkeit, umfassendere Leistungskriterien aufzustellen. Dabei zeigte sich, daß die für den Stapelbetrieb bereits eingeführten Bewertungsmaßstäbe [KUE72,HEL74] wie Durchsatz, mittlere Verweilzeit etc. für Prozeßrechner nur beschränkt anwendbar sind. Es wurden daher Versuche unternommen, unter Berücksichtigung der für ein Prozeßautomatisierungssystem zusätzlich relevanten Bedingungen, wie z.B. Zuverlässigkeit und Wartbarkeit, anwendungsunabhängige Leistungsmerkmale zu bestimmen, die eine anwendungsabhängige Leistungsbeurteilung erlauben sollen [LAU74,VDI76].

Von besonderem Interesse für diese Arbeit sind jene Kriterien, die sich auf die Erfüllung von Zeitbedingungen beziehen. Da in heutigen Prozeßrechnern meist eine Vielzahl von einander mehr oder weniger unabhängigen Vorgängen parallel bzw. quasi-parallel abläuft, und demnach auch unterschiedliche Zeitbedingungen vorliegen, reicht die erzielbare Reaktionszeit als alleinige Bewertungsgrundlage nicht aus. Parallele Vorgänge (Tasks, Rechenprozesse) werden durch Betriebssysteme verwaltet und koordiniert, was zur Folge hat, daß die Fähigkeit dieser Verwaltungsfunktionen, mit den gestellten Anforderungen fertig zu werden, im Vordergrund steht. Dabei handelt es sich hauptsächlich um die Forderungen nach Rechtzeitigkeit und Gleichzeitigkeit [LAU76].

In den folgenden Abschnitten wird versucht, diese Kriterien näher zu klassifizieren.

#### 2.4. Klassifikation von zeitlichen Anforderungen

Von den für einen Prozeßrechner relevanten Zeitbedingungen sind - bezogen auf das gesamte Automatisierungssystem - zwar nur jene von Interesse, die sich auf die Kommunikationsvorgänge zwischen dem Rechner und seiner Prozeßumgebung beziehen (z.B. rechtzeitiges Betätigen einer Alarmglocke oder gleichzeitiges Abholen von einander entsprechenden Meßwerten), rechnerintern überträgt sich dies jedoch auch auf die Wechselwirkungen zwischen einzelnen Tasks und ihrer Umgebung (z.B. Datensätze oder andere Rechenprozesse).

Letzteres ist der allgemeine Fall, da der technische Prozeß auch als Teil der Umgebung einzelner Tasks betrachtet werden kann. Demnach können also zeitliche Anforderungen an einen Prozeßrechner einheitlich als jene (Zeit-)Bedingungen aufgefaßt werden, denen die vorgesehenen Einflußnahmen der Rechenprozesse auf ihre Umgebung unterworfen sind. In diesem Sinne bietet sich eine Einteilung in zwei große Kategorien an:

- A - "Absolute Zeitbedingungen", d.h. Bedingungen, die bei ihrer Vereinbarung (!) bezogen auf die Zeitachse festliegen
- B - "Relative Zeitbedingungen", d.h. Bedingungen, die relativ zu noch bevorstehenden (quasi-) stochastischen Ereignissen angegeben werden, die von der Umgebung auf die betreffenden Tasks einwirken, oder von diesen selbst erzeugt werden.

Beispiele für die Kategorie A sind die Angaben 'Um 10 Uhr  $\pm$  1 Minute', 'Ab 12 Uhr alle 2 Minuten', 'Spätestens nach 2 Sekunden'.

Ein Beispiel für die Kategorie B ist die Angabe 'Innerhalb von 1 Sekunde nach Eintreffen des Alarms "Behälter voll"', aber auch die Forderung '5 Sekunden nach erfolgtem Start (durch eine andere Task)'.

Die Anforderungen der Kategorie A können auch wie folgt formuliert werden:

Es ist - zum Zeitpunkt der Vereinbarung der Bedingung ( $t_v$ )- ein Zeitintervall (oder mehrere Zeitintervalle)  $[t_1', t_1'']$  definiert, in dem (in denen) die betreffende Einflußnahme einer Task auf ihre Umgebung zu erfolgen hat.

Analog gilt für die Kategorie B:

Es ist - zum Zeitpunkt ( $t_e$ ) eines bevorstehenden Ereignisses - ein Zeitintervall (oder mehrere Zeitintervalle)  $t_e + \Delta t_i', t_e + \Delta t_i''$  definiert, in dem (in denen) diese Einflußnahme erfolgen soll.

In der Folge werden aus Gründen der einheitlichen Darstellung die  $t_e + \Delta t_i$  ebenfalls mit  $t_i$  bezeichnet.

Abhängig von Lage und Größe der genannten Zeitintervalle läßt sich eine weitere Einteilung in 4 Gruppen vornehmen:

a -  $t_i' = t_i - \xi' < t_i + \xi'' = t_i''$

d.h. es existieren Zeitpunkte  $t_i$  mit den Toleranzen  $\xi'$  und  $\xi''$  nach 'unten' bzw. 'oben', zu denen die Einflußnahme erfolgen soll. (Ein wiederholtes Auftreten der Zeitbedingung - z.B. zyklisch - ist möglich und sinnvoll)

b -  $t_i' = t_i'' = t_i$

d.h. die Einflußnahme hat genau zu den Zeitpunkten  $t_i$  zu erfolgen. (Ein wiederholtes Auftreten der Zeitbedingung ist hier ebenfalls sinnvoll)

c -  $t_i' = t_e$  (bzw.  $t_v$ ),  $t_i'' > t_e$  (bzw.  $t_v$ )

diese Alternative besagt, daß die Einflußnahme spätestens zum Zeitpunkt  $t_i''$  nach Eintreffen des Ereignisses (bzw. nach der Vereinbarung) zu erfolgen hat.

(Da ein mehrfaches Auftreten der Zeitschranke  $t_i''$  hier nicht sinnvoll ist, wurde der Index i durch die Zahl 1 ersetzt)

d -  $t_i' > t_e$  (bzw.  $t_v$ ),  $t_i'' \rightarrow \infty$

d.h. die Einflußnahme darf frühestens zum Zeitpunkt  $t_i'$  erfolgen.

(Ein mehrfaches Auftreten dieser Zeitbedingung ist nur dann sinnvoll, wenn auf Grund der vorliegenden (geringen) Rechnerauslastung die frühestens zum Zeitpunkt  $t_i'$  vorgesehene Aktion mit großer Wahrscheinlichkeit in dem Intervall  $[t_i', t_{i+1}']$  erfolgen kann -

andernfalls tritt u.U. eine unzulässige Häufung von Einplanungen auf)

Streng genommen sind die Alternativen b bis d nur Sonderfälle von a. Die Unterscheidung wurde hier deshalb vorgenommen, weil sich dadurch die Klassifizierung der praktischen Anwendungsfälle einerseits, sowie die Einstufung der Leistungsfähigkeit von Taskverwaltungsstrategien andererseits vereinfacht.

Insbesondere stellt der Fall b "eigentlich" keine eigenständige Alternative dar, da die genaue Einhaltung von Sollzeitpunkten in der Praxis ja nicht zu erzielen ist, und demnach Fall a zur Anwendung käme. Hier sollen jedoch Intervalle, die klein gegenüber den Ausführungszeiten von üblichen Taskverwaltungsfunktionen sind, als "Zeitpunkte" betrachtet werden.

In Bild 2.2 sind die Alternativen a bis d noch einmal graphisch dargestellt.

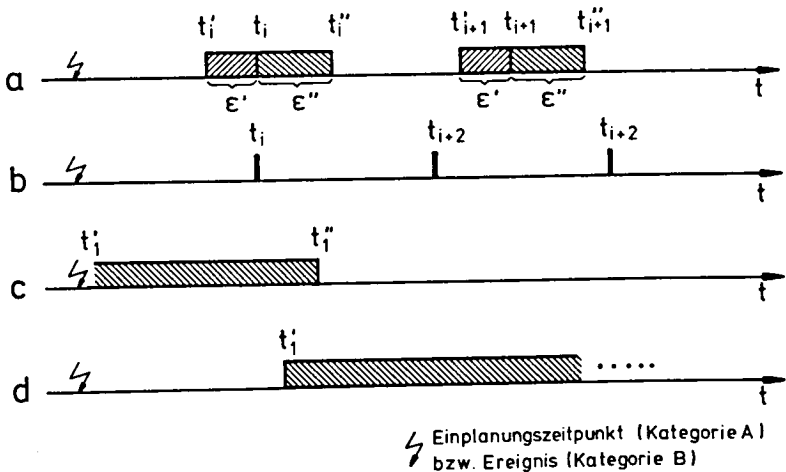


Bild 2.2: Klassifizierung der "Sollzeitintervalle" bei der Angabe von Zeitbedingungen

## 2.5. Praktische Beispiele

In der Folge wird an Hand einiger Beispiele gezeigt, wie sich die genannten Alternativen von Zeitbedingungen in der Praxis darstellen.

### 2.5.1. Walzstraßenautomatisierung

In Bild 2.3 ist eine Kaltwalzstraße schematisch dargestellt. Es handelt sich dabei um einen zeitkritischen Fließprozeß.

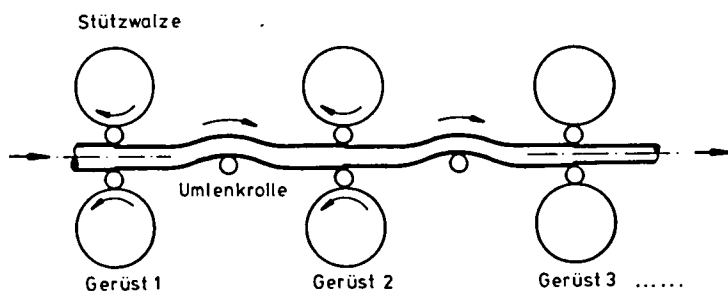


Bild 2.3: Schema einer Kaltwalzstraße

Den gewünschten Verformungen des Walzgutes an den einzelnen Gerüsten liegt ein mathematisches Modell zugrunde, dessen Parameter in erster Linie von der Materialbeschaffenheit abhängen. Da diese Materialbeschaffenheit (Dicke, Festigkeit, etc.) im Normalfall aber nicht genau genug vorherbestimmbar ist und außerdem Schwankungen unterworfen ist, muß während des Walzvorganges ständig eine Vielzahl von Meßwerten aufgenommen werden, die zur ständigen Anpassung des Modells und damit zur Veränderung von Stellgrößen (z.B. Position der Umlenkrollen, Dicke der Walzspalte etc.) herangezogen werden.

Zu diesen Meßwerten gehört unter anderem die Biegekraft, die Walzkraft und das Walzmoment an den Gerüsten, die Blechdicke zwischen den einzelnen Gerüsten, sowie die Zugkraft an den Umlenkrollen.

Durch jede (analoge) Messung wird gleichsam ein Stück Blech von einer durch die Integrationszeit bestimmten Länge als "eine Stelle" des

Walzgutes betrachtet. Um die Umformungsvorgänge möglichst exakt bestimmen zu können, wird versucht, eine derartige "Stelle" über die gesamte Straße zu verfolgen und die Messungen jeweils dann vorzunehmen, wenn diese Stelle die jeweiligen Geber passiert. Für diese Verfolgung können z.B. Zähler verwendet werden, die in Abhängigkeit von der Umdrehungsgeschwindigkeit der Walzen zu den betreffenden Zeitpunkten Unterbrechungssignale an den Prozeßrechner liefern. Es treten bei diesem Automatisierungsproblem zwei unterschiedliche Zeitengpässe auf: Erstens sollen möglichst viele Meßwerte aufeinanderfolgen, d.h. die gemessenen "Stellen" des Walzgutes sollen kurze Abstände haben, und zweitens sollen die verschiedenen Messungen beim Durchlauf durch die Walzstraße nach Möglichkeit immer an denselben Stellen des Bleches vorgenommen werden. Die letztgenannte Forderung ist am schwersten zu erfüllen: Nimmt man z.B. an, daß das Walzgut eine Geschwindigkeit von etwa 15m/s hat und daß eine Messung 20 ms dauert, so ergibt sich die Länge eines gemessenen Blechstückes zu 30 cm. Soll nun dieses Blechstück mit einer maximalen Abweichung von  $\pm 6$  cm an den verschiedenen Gerüsten erfaßt werden, so müßten die einzelnen Meßzeitpunkte einer Genauigkeit von  $\pm 4$  ms genügen.

Verwendet man nun für die Verfolgung des Materials die oben erwähnten Zähler und Unterbrechungssignale, so sind diese Zeitbedingungen der Kategorie 'B' (relativ), Gruppe 'a' (Sollzeitpunkte mit Toleranzen) zuzuordnen. Analog dazu handelt es sich um 'absolute' Zeitbedingungen (Kategorie 'A'), wenn die Berechnung der Sollzeitpunkte durch den Rechner selbst über Geschwindigkeitsmessungen erfolgt. Eine (nicht befriedigende) Lösung der Aufgabe mit Hilfe der Programmiersprache PEARL könnte dann wie folgt aussehen (stark vereinfacht):

Zeile

```
1  MESSUNG : TASK /* Task zu Erfassung der Meßwerte*/
2      FOR I TO GERUESTANZAHL REPEAT /* für alle Gerüste*/
3          TAKE FROM VMESS (I) TO V; /* Geschwindigkeit messen */
4          TAKE FROM DMESS (I) TO D (I); /* Dicke messen */
5          AFTER GERUESTABSTAND/V*1.00 SEC RESUME;
           /* Warten, bis Meßstelle an nächstem Gerüst */
6      END;
7  END;
```



Dabei ergeben sich die folgenden Schwierigkeiten:

- . Zwischen der Messung der Blechdicke am Gerüst  $i$  (Zeile 4) und der Einplanung der Messung an Gerüst  $i+1$  (Zeile 5) vergeht ein im Prinzip unbekanntes Zeitintervall, das aber bei der Einplanung berücksichtigt werden müßte.
- . Die Einplanung in Zeile 5 betrifft de facto nicht die Messung, sondern nur die Fortsetzung der Task. Bis zur eigentlichen Messung vergeht wieder ein unbekanntes Zeitintervall.

Es ist also ersichtlich, daß die eigentlichen Anforderungen hier nicht zufriedenstellend formuliert sind, wobei bemerkt werden muß, daß dies mit den heute üblichen Sprachmitteln in diesem konkreten Fall grundsätzlich nicht möglich sein dürfte. Dabei ist die Frage außer acht gelassen, ob eine Taskverwaltungsstrategie mit den angenommenen Toleranzen von 4 Millisekunden überhaupt fertig werden könnte.

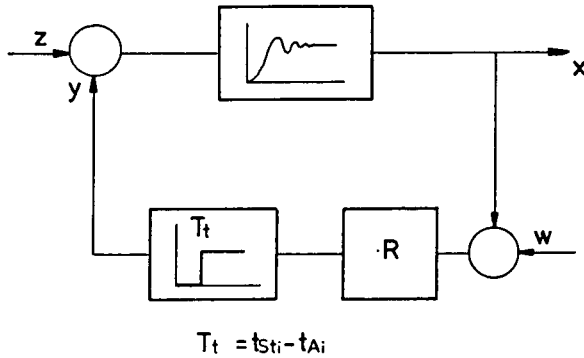
In der Praxis wird solch harten Bedingungen nach wie vor mit Assemblerprogrammierung begegnet, bei der eine "Zwangssequentialisierung" der relevanten Befehlsfolgen und damit die Berechnung von resultierenden Ausführungszeiten einfacher ist als bei üblichen höheren Programmiersprachen.

#### 2.5.2. Direkte Digitale Computerregelung (DDC)

Bei einigen Regelalgorithmen, die eine relativ schnelle Angleichung der Regelgröße an den Sollwert bewirken (z.B. "deadbeat-response", "halbproportionaler" Algorithmus), sollten die Abtastzeitpunkte, bzw. Stellzeitpunkte möglichst genau eingehalten werden, da sonst der gewünschte Effekt verlorenght.

In der Praxis hauptsächlich verwendete, "langsamere" Algorithmen gehen vielfach von der Gleichzeitigkeit von Abtastung und Stellgrößenbeeinflussung aus. Bei zeitkritischeren Verfahren muß aber die Ausführungszeit des Regelalgorithmus berücksichtigt werden, was üblicherweise dadurch geschieht, daß bei der Synthese des Regelkreises ein Totzeitglied eingeführt wird, dessen Totzeit gleich der Differenz zwischen Stellzeitpunkt und Abtastzeitpunkt ist, wobei letztere wieder ausreichend größer als die Ausführungszeit des Algorithmus sein muß, um den Rechner nicht zu stark auszulasten

(siehe Bild 2.4).



$x$ ... Regelgröße	$T_t$ ... Totzeit
$y$ ... Stellgröße	$t_{Sti}$ ... Stellzeitpunkte
$z$ ... Störungsgröße	$t_{Ai}$ ... Abtastzeitpunkte
$w$ ... Führungsgröße	$R$ ... Regler

Bild 2.4: Berücksichtigung der Zeitdifferenz zwischen Abtastung und Stellgrößenbeeinflussung durch ein Totzeitglied

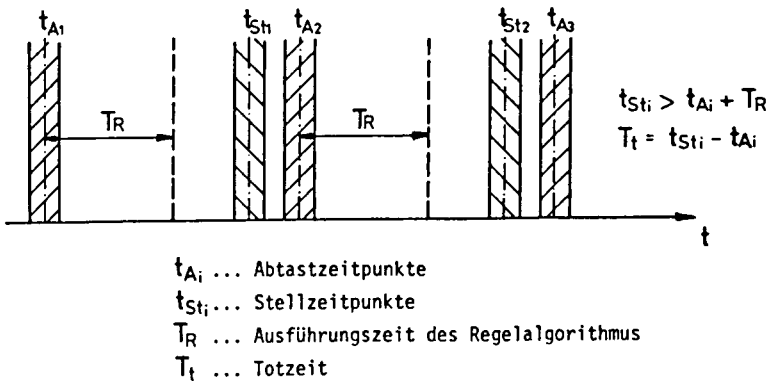


Bild 2.5: Sollzeitintervalle für Abtastung und Stellgrößenbeeinflussung

In Bild 2.5 ist ein mögliches Zeitschema skizziert.

Es handelt sich bei diesem Beispiel um Zeitprobleme der Kategorie "Aa" (absolut - mit Toleranz), wobei die Toleranzen üblicherweise so groß sind, daß ein Taskverwaltungsmechanismus für quasi-parallele Abarbeitung ohne weiteres zu ihrer Einhaltung ausreicht. Die exakte Formulierung des Problems ist jedoch auch hier mit herkömmlichen Sprachmitteln nicht möglich. Deshalb liegt es nahe, Betriebsmittelzuteilungsstrategien (bzw. Sprachmittel) zu fordern, die es ermöglichen, diesen Zeitbedingungen unmittelbar Rechnung zu tragen. Bei konventionellen Prioritätszahlen könnte dies ersatzweise dadurch geschehen, daß den betreffenden Automatisierungsprogrammen Prioritäten zugeteilt werden, die proportional zu den entsprechenden Abtastzyklen sind. In [SER72] wird gezeigt, daß eine Einhaltung aller Zeitbedingungen bei einer großen Anzahl zyklischer Tasks mit dieser Methode nur dann gewährleistet werden kann, wenn die Rechnerauslastung unter 70% liegt. Im Gegensatz dazu erlauben modernere Strategien eine Auslastung bis 100%. Diese Zahlen gelten allerdings nur unter relativ einfachen Randbedingungen - z.B. bei ausschließlicher Konkurrenz um einen Prozessor.

### 2.5.3. "Zielbremssteuerung"

Bei einer Zielbremssteuerung handelt es sich um einen sequentiellen Vorgang, der üblicherweise Bestandteil eines übergeordneten Stückprozesses ist.

Als Beispiel sei hier die Ansteuerung der Positionen eines Hochregallagers erwähnt:

Die Reihenfolge der notwendigen Aktivitäten eines Steuerprogramms, das ein Regalförderzeug an einen gewünschten Zielort bringt, könnte wie folgt aussehen:

- 1 Übernahme der Sollkoordinaten
- 2 Auf Grund der Soll-Ist-Wertdifferenz Motor ansteuern (rechts-links, oben-unten)
- 3 Zählung der durch Vorbeifahren an Zwischenpositionen erzeugten Unterbrechungssignale
- 4 Rechtzeitiges Einschalten der Bremsen

## 5 Überprüfung der Zielposition und eventuell Korrektur durch Langsamfahrt

Je genauer der unter 4 erwähnte Bremszeitpunkt eingehalten wird, desto eher können Pendelungen, wie sie unter 5 angedeutet sind, unterbleiben. Nimmt man an, daß die Geschwindigkeit eines Regalförderzeuges etwa 1m/sec beträgt und daß die Zielposition auf  $\pm 1\text{cm}$  genau angesteuert werden muß, so ergibt sich eine Toleranz für den Bremszeitpunkt von  $\pm 10$  Millisekunden. Der zeitliche Vorgang ist in Bild 2.6 schematisch dargestellt.

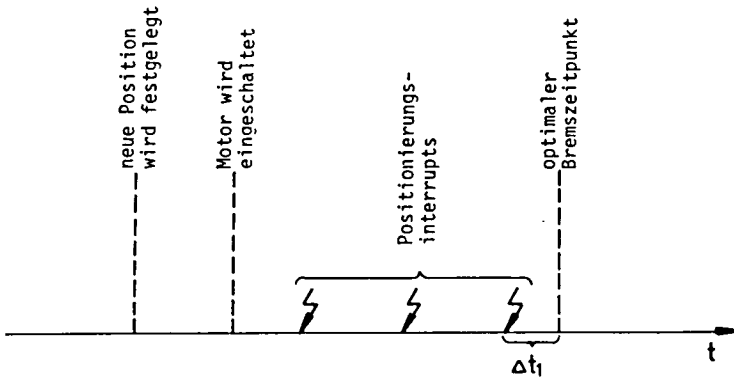


Bild 2.6: Zeitlicher Ablauf der Ansteuerung einer Hochregallagerposition

Dieses Beispiel fällt unter die Kategorie "Ba" (relativ - mit Toleranz).

### 2.5.4. Dynamische Prüfung [GHA77]

Bei dynamischen Prüfungsvorgängen tritt häufig der Fall auf, daß nach Veränderung eines Zustandes des Prüflings bestimmte Werte innerhalb einer vorgegebenen Zeitspanne definierte Sollwerte erreicht haben sollen. Dies kann z.B. über eine Nachlaufregelung geschehen. In Bild 2.7 sei das ideale Zeitverhalten einer Regelgröße  $x$  darge-

stellt. In einem vorgegebenen Zeitabstand vom Zeitpunkt der Zustandsänderung werden einige Kontrollmessungen zum Soll-Ist-Wert-Vergleich vorgesehen, die innerhalb vorgegebener Toleranzen erfolgen müssen.

Wird die Zustandsänderung dem Rechner über einen Interrupt gemeldet, so handelt es sich nach der Definition in Kap.2.4. um Zeitbedingungen der Kategorie "Ba" (relativ - mit Toleranz). Falls jedoch der Rechner selbst die Zustandsänderung vornimmt, ergibt sich ein zusätzliches Problem: Da die zustandsverändernde Aktion (z.B. 'Neuen Sollwert vorgeben') und die Einplanung der Messung nicht gleichzeitig erfolgen kann, sind verfälschende Zeitverschiebungen denkbar. Die maßgeblichen Aktionen seien beispielsweise:

Schritt 1: Neuen Sollwert vorgeben

Schritt 2: 'Führe Schritt 3 nach  $10(+0.1)$ sec aus'

Schritt 3: Kontrollmessung

Falls in der angegebenen Reihenfolge vorgegangen wird, so kann die Messung trotz Einhaltung der vorgegebenen Toleranzen zu spät erfolgen, da zwischen den Schritten 1 und 2 eine unbekannte Zeit verstreicht. Analog dazu kann bei der Reihenfolge 2-1-3 die Messung zu früh erfolgen.

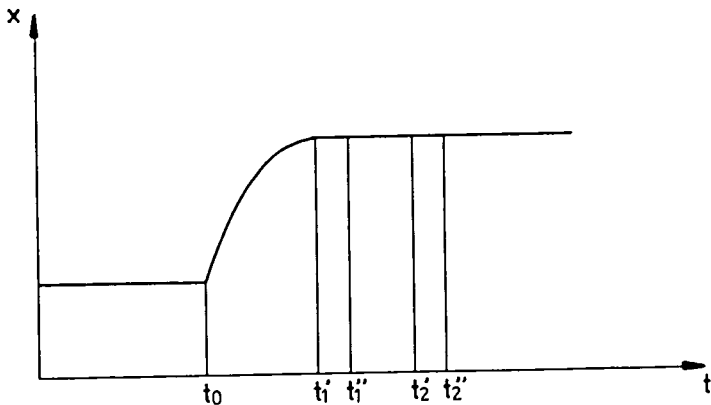


Bild 2.7: Ideales Verhalten der Regelgröße  $x$  bei einer Nachlaufregelung

Nun könnte man denken, daß eine zusätzliche Angabe von Zeitbedingungen für die zustandsverändernde Aktion das Problem löst. Dies ist aber nur der Fall, wenn dann beide Einplanungen (für die Sollwertvorgabe und die Messung) eine "unteilbare" - also "gleichzeitige" Aktion darstellen, etwa in der folgenden Form:

Schritt 1: 'Führe Schritt 2 innerhalb von 0.1sec und  
Schritt 3 nach 10(+0.1)sec aus'

Schritt 2: Neuen Sollwert vorgeben

Schritt 3: Kontrollmessung

Da eine derartige "unteilbare" Aktion in der Praxis jedoch nicht so ohne weiteres zu realisieren ist, scheint der folgende Ansatz die beste Lösung darzustellen:

Schritt 1: 'Führe die folgenden Aktionen bis einschließlich Schritt 3 innerhalb von 0.1sec aus'

Schritt 2: Neuen Sollwert vorgeben

Schritt 3: 'Führe Schritt 4 nach 10(+0.1)sec aus'

Schritt 4: Kontrollmessung

Es muß bemerkt werden, daß bei Einhaltung aller Zeitbedingungen der zeitliche Abstand zwischen Sollwertvorgabe und Messung bei dieser Lösung zwischen 9.9 und 10.2 Sekunden liegen kann.

Eine detaillierte Diskussion der Zeitprobleme bei Folgeprozessen findet man in [GHA77].

#### 2.5.5. Meßwertüberwachung auf gleitende Grenzen

Die Zeitprobleme bei einer Meßwertüberwachung auf gleitende Grenzen sind den in Kap.2.5.4. beschriebenen ähnlich. In Bild 2.8 ist der Verlauf einer zu überwachenden Prozeßgröße mit den vorgegebenen Grenzwerten dargestellt.

Für den Fall, daß die Meßzeitpunkte  $t_i$  nicht genau eingehalten werden, z.B. weil sie innerhalb von vorgegebenen Toleranzen schwanken können, besteht die Möglichkeit, daß noch einwandfreie Verläufe als fehlerhaft bzw. bereits fehlerhafte als einwandfrei erkannt

werden. Dies ist in Bild 2.8 durch die schwarzen Dreiecke symbolisiert. Eine Möglichkeit wäre nun die Einführung von 2 parallelen "Ersatzgrenzen" mit der Bedeutung 'Grenzwert möglicherweise überschritten' und 'Grenzwert sicher überschritten'. In der Praxis ist dieses Problem jedoch nicht sehr gravierend, da wegen eines meist relativ schwachen Anstiegs der betreffenden Kurven aus durchaus einhaltbaren Zeittoleranzen vernachlässigbare Schwankungsbreiten bei den Grenzwerten entstehen.

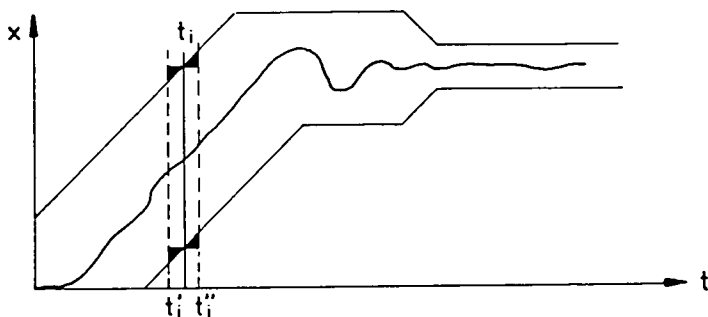


Bild 2.8: Meßwertüberwachung auf gleitende Grenzen

#### 2.5.6. Prüfstände

Bei Prüfständen für Aggregate, deren Zeitverhalten große Dynamik aufweist (z.B. Motorprüfstände), kommt es beispielsweise bei der Aufnahme von Kennlinien darauf an, daß einander zugeordnete Meßwerte möglichst zum gleichen Zeitpunkt aufgenommen werden, da sonst Fehlschlüsse gezogen würden. Es handelt sich hier meist um derart geringe zulässige Abweichungen, daß dieses Problem unter die Alternative "b" (Sollzeitpunkte ohne Toleranzen) fällt. Je nachdem, ob ein äußeres Ereignis (z.B. Zündzeitpunkt) oder ein rechnerinterner Algorithmus die Meßzeitpunkte bestimmt, handelt es sich nach der oben eingeführten Klassifikation um "relative" oder "absolute" Zeitbedingungen.

Auch hier gilt wieder die unter 2.5.1. gemachte Aussage, nämlich daß man extrem harten zeitlichen Anforderungen nicht mehr über quasi-parallele Mechanismen sondern nur über eine Sequentiali-

sierung, bzw. im konkreten Fall über eine zwangsweise Kopplung der notwendigen Befehlsabläufe gerecht werden kann.

#### 2.5.7. Ablaufberg

Eines der Detailprobleme bei der Automatisierung eines Ablaufberges ist das rechtzeitige Verstellen der Weichen. Man kann z.B. über Messungen der Waggongeschwindigkeiten und geeignete Signale, die durch überfahrene Kontakte ausgelöst werden, die jeweils spätesten Schaltzeitpunkte bestimmen. In diesem Fall handelt es sich um Zeitbedingungen der Kategorie "Bc" (relativ zum Ereignis 'Kontakt überfahren' - Reaktion spätestens nach Verstreichen eines vorgegebenen Intervalls). Falls noch darauf zu achten ist, daß ein etwaiger Vorgänger die zu schaltende Weiche bereits überfahren hat, kommt eine Zeitbedingung der Kategorie "Bd" (relativ zum Ereignis 'Weiche überfahren' - Reaktion frühestens nach Verstreichen eines vorgegebenen Intervalls) dazu. In beiden Fällen handelt es sich um Angaben der Größenordnung Sekunden.

Auf den ersten Blick scheint die Lösung dieses Problems nicht schwierig zu sein, doch können durch eine große Anzahl von Wagen und Weichen auch großzügig bemessene Zeittoleranzen zu Engpässen führen.

#### 2.5.8 Anfahrvorgänge

Anfahrvorgänge sind typische Folgeprozesse. Es werden über einzelne Steuerschritte Zustandsänderungen im Prozeß bewirkt, die abschließend (meist) zu einem stationären Zustand führen. In Bild 2.9 ist die Sequenz zweier Steuerschritte beim Einschalten des Ölbrenners eines Kraftwerkblockes dargestellt [LAU76].

Bei diesem Beispiel kommt es darauf an, mindestens 15 Sekunden Abstand zwischen den beiden Steuerschritten einzuhalten. Da die obere Schranke von sekundärer Bedeutung ist, brauchen dafür üblicherweise keine Angaben gemacht zu werden, sofern nicht eine zu starke Rechnerauslastung unzulässige Verzögerungen hervorrufen würde. Es handelt sich demnach bei diesem Beispiel um Zeitbedingungen der Kategorie "Ad" (absolut - Aktion frühestens zu einem vorgegebenem Zeitpunkt).



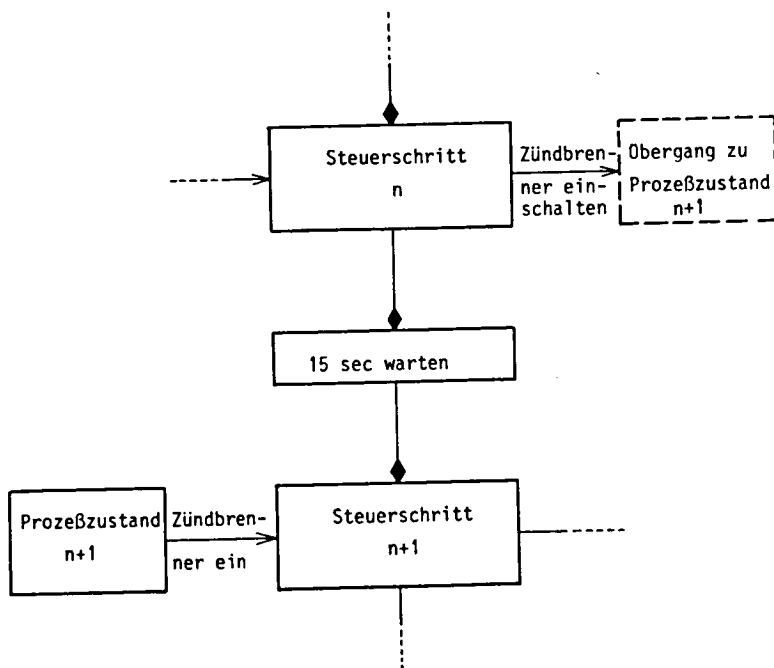


Bild 2.9: Steuerschritte beim Einschalten eines Ölbrenners

## 2.6. Lösungswege

Die Frage, ob die in den vorangegangenen Kapiteln beschriebenen Zeitprobleme durch Softwaremaßnahmen gelöst werden können, muß in zwei Stufen beantwortet werden:

Zum ersten kommt es darauf an, ob eine Anforderung überhaupt beschrieben werden kann, d.h. ob der Programmierer die Möglichkeit hat, die Zeitbedingungen in irgendeiner Form zum Ausdruck zu bringen. Zum zweiten gilt es festzustellen, in welchem Maße das Rechensystem (Hardware und Software) den beschriebenen Anforderungen gerecht werden kann.

Leider verhält es sich vielfach so, daß jeweils nur ein Teil der Frage zufriedenstellend beantwortet werden kann. Beispielsweise ist es in verschiedenen Sprachen zwar möglich, die Ausführung einer Aktion zu einem bestimmten Zeitpunkt zu vereinbaren (z.B. 'AT 10:00 ACTIVATE MESSUNG'); wann sie jedoch tatsächlich stattfindet, hängt von Umständen ab, auf die der Programmierer nur sehr indirekt Einfluß hat.

Andererseits kann die Forderung, daß etwaspätestens 2 Sekunden nach Eintreffen eines Alarmsignals zu geschehen habe, in gängigen Systemen zwar nicht formuliert, aber über geeignete Maßnahmen (z.B. Prioritätsfestlegung) meist eingehalten werden.

In den Bildern 2.10 bis 2.13 sind die im Kapitel 2.4 klassifizierten Zeitbedingungen in Matrixform dargestellt. Die im Kapitel 2.5 angeführten Beispiele sind in die entsprechenden Felder eingeordnet. Eine durchgehende Schraffierung bringt zum Ausdruck, daß die betreffenden Zeitprobleme zufriedenstellend zu behandeln sind. Eine strichlierte Schraffierung läßt diese Aussage nur mit Einschränkung zu.

In Bild 2.10 sind die Beschreibbarkeit der Zeitprobleme mit herkömmlichen Sprachmitteln (Prioritätszahlen) dargestellt. Der Umstand, daß keine Unterscheidung zwischen den Alternativen 'b' und 'd' möglich ist, wird etwas vereinfachend dahingehend ausgelegt, daß beide beschrieben werden können.

Bild 2.11 zeigt im Gegensatz dazu die Lösbarkeit der Zeitprobleme mit Strategien, denen vom Anwender angegebene Prioritätszahlen zugrunde liegen. Man sieht, daß die Alternative 'b' überhaupt nicht, die Alternativen 'a' und 'c' nur mit Einschränkungen bewältigt werden können.

Die Bilder 2.12 und 2.13 zeigen die Verhältnisse, wie sie sich bei der Einführung von Zeitschranken als Sprachmittel bzw. von zeitschrankengesteuerten Taskverwaltungsstrategien darstellen.

Man sieht, daß auch mit diesen Strategien eine Lösung der Zeitprobleme der Gruppe 'b' praktisch unmöglich ist. Die Gruppen 'a' und 'c' können jedoch - im Vergleich zu konventionellen Techniken - besser behandelt werden. Voraussetzung dafür ist allerdings, daß die vorgegebenen Toleranzintervalle erheblich größer als die bei der betreffenden Strategie auftretenden Verwaltungszeiten für die

Zeit- bedingungen	A 'absolut'	B 'relativ'
<sup>a</sup> $t_i = t_i - \varepsilon$ $t_i + \varepsilon = t_i'$	Walzstraße DDC Gleitende Grenzen	Walzstraße Zielbremssteuerung Dynamische Prüfung
<sup>b</sup> $t_i = t_i' = t_i$	Prüfstände	Prüfstände
<sup>c</sup> $t_i = \left\{ \begin{matrix} t_e \\ t_v \end{matrix} \right\}$ $t_i' > t_i$		Ablaufberg
<sup>d</sup> $t_i > \left\{ \begin{matrix} t_e \\ t_v \end{matrix} \right\}$ $t_i' \rightarrow \infty$	Anfahrvorgänge Chargenprozesse	Anfahrvorgänge Chargenprozesse

Bild 2.10: Beschreibbarkeit der Zeitprobleme mit herkömmlichen Sprachmitteln (Prioritätszahlen)

Zeit- bedingungen	A 'absolut'	B 'relativ'
<sup>a</sup> $t_i = t_i - \varepsilon$ $t_i + \varepsilon = t_i'$	Walzstraße DDC Gleitende Grenzen	Walzstraße Zielbremssteuerung Dynamische Prüfung
<sup>b</sup> $t_i = t_i' = t_i$	Prüfstände	Prüfstände
<sup>c</sup> $t_i = \left\{ \begin{matrix} t_e \\ t_v \end{matrix} \right\}$ $t_i' > t_i$		Ablaufberg
<sup>d</sup> $t_i > \left\{ \begin{matrix} t_e \\ t_v \end{matrix} \right\}$ $t_i' \rightarrow \infty$	Anfahrvorgänge Chargenprozesse	Anfahrvorgänge Chargenprozesse

Bild 2.11: Lösbarkeit der Zeitprobleme mit herkömmlichen Strategien (Prioritätszahlen)

Zeit- bedingungen	A 'absolut'	B 'relativ'
a $t_i = t_i - \varepsilon'$ $t_i + \varepsilon' = t_i'$	Walzstraße DDC Gleitende Grenzen	Walzstraße Zielbremssteuerung Dynamische Prüfung
b $t_i = t_i' = t_i$	Prüfstände	Prüfstände
c $t_i = \begin{Bmatrix} t_e \\ t_v \end{Bmatrix}$ $t_i' > t_i$		Ablaufberg
d $t_i > \begin{Bmatrix} t_e \\ t_v \end{Bmatrix}$ $t_i' \rightarrow \infty$	Anfahrvorgänge Chargenprozesse	Anfahrvorgänge Chargenprozesse

Bild 2.12: Beschreibbarkeit der Zeitprobleme bei der Einführung von Zeitschranken als Sprachmittel

Zeit- bedingungen	A 'absolut'	B 'relativ'
a $t_i = t_i - \varepsilon'$ $t_i + \varepsilon' = t_i'$	Walzstraße DDC Gleitende Grenzen	Walzstraße Zielbremssteuerung Dynamische Prüfung
b $t_i = t_i' = t_i$	Prüfstände	Prüfstände
c $t_i = \begin{Bmatrix} t_e \\ t_v \end{Bmatrix}$ $t_i' > t_i$		Ablaufberg
d $t_i > \begin{Bmatrix} t_e \\ t_v \end{Bmatrix}$ $t_i' \rightarrow \infty$	Anfahrvorgänge Chargenprozesse	Anfahrvorgänge Chargenprozesse

Bild 2.13: Lösbarkeit der Zeitprobleme mit zeitschranken-  
gesteuerten Strategien

Taskkoordinierung sind. Letztere bewegen sich bei heutigen Prozeßrechnern erfahrungsgemäß in der Größenordnung von 100 bis 1000 Mikrosekunden (bei hauptspeicherresidenten Systemen). Für die kleinsten zulässigen Zeittoleranzen sollte mindestens das jeweils zehnfache veranschlagt werden.

Bei kleineren Spielräumen bzw. bei Zeitbedingungen der Gruppe 'b' (Zeitpunkte) scheitern im Prinzip alle Strategien, die auf Prozessor-multiplexen beruhen.

Handelt es sich um Systeme, bei denen nur absolute Zeitbedingungen zur Anwendung kommen, und bei denen auch die Laufzeiten aller beteiligten Rechenprozesse bekannt sind, so können die gesamten Aktivitäten (ev. durch komplexe Strategien) in ihrem Ablauf vorherbestimmt werden. Dies bedeutet, daß das gesamte Programmsystem als ein rein sequentielles Programm betrachtet werden kann, und daß sich demzufolge Taskverwaltungsstrategien erübrigen. Sobald jedoch stochastische Einflüsse in irgendeiner Form in das System eingehen (und das ist bei der Prozeßautomatisierung praktisch immer der Fall), sind solche Maßnahmen nicht mehr möglich.

Wenn nur ein kleiner (leicht überschaubarer) Teil des Gesamtprozesses solch harten Zeitbedingungen ausgesetzt ist, bleibt die Möglichkeit, die zugehörigen Rechenprozesse auf 'privilegierter' Ebene abzuwickeln, d.h. der Kontrolle der Taskverwaltungsmechanismen zu entziehen. Die dann entstehenden Antwortzeiten lassen sich in gewissem Rahmen vorherberechnen und ermöglichen dadurch die Lösung von zeitkritischeren Problemen, als dies auf 'Anwenderebene' möglich ist. Die Nachteile dieser Vorgangsweise wurden bereits in den Kapiteln 2.1. und 2.2. dargelegt. Wenn auch dies nicht den gewünschten Effekt nach sich zieht, bleibt als letzte Konsequenz nur mehr eine geeignete Dezentralisierung, d.h. eine Delegation von zeitkritischen Einzelaufgaben an 'periphere Intelligenzen', was ja auch aus anderen Gründen (z.B. Zuverlässigkeit) vorteilhaft erscheint und sich durch die rasante Entwicklung auf dem Gebiet der Mikroprozessortechnik immer mehr durchzusetzen beginnt.

### 3. TASKVERWALTUNGSSTRATEGIEN FÜR PROZESSRECHNER

Wie schon eingangs erwähnt, ist nur ein Teil der Strategien, die im Stapelbetrieb zum Einsatz kommen, auch für Prozeßrechner geeignet. In der Folge wird eine Anzahl solcher Strategien vorgestellt, die entweder bei Realzeitanwendungen bereits weit verbreitet sind, oder in der Literatur im Zusammenhang mit Problemen der Realzeitanwendungen diskutiert werden.

Im Anschluß daran wird eine Erweiterung einer dieser Strategien beschrieben, die für bestimmte Anwendungsfälle nachweislich zu einer Verbesserung des Zeitverhaltens führt.

Zusätzlich wird ein Konzept erläutert, das auf die besonderen Verhältnisse beim Auftreten sogenannter "kritischer Abschnitte" zugeschnitten ist und das zusätzlich zu den beschriebenen Strategien eingesetzt werden kann.

Abschließend werden Implementations- und Anwendungsaspekte diskutiert.

#### 3.1. Beschreibung der Strategien

Die "Priorität"  $p_i$  einer Task  $T_i$  sei allgemein als Größe definiert, die in einer Konkurrenzsituation um ein Betriebsmittel bestimmt, an welche Position der betreffenden Warteschlange die Task eingereiht wird, wobei die Task mit der höchsten Priorität sich an der vordersten Stelle befindet. In Ergänzung dazu sei unter der "Prioritätszahl"  $\overline{p}_i$  eine der Task  $T_i$  zugeordnete Ganzzahl zu verstehen, die umso kleiner ist, je höher die zugehörige Priorität ist.

Damit kann man Taskverwaltungsstrategien nach den folgenden Kriterien abstufen:

- a. Die Prioritätszahlen und damit die relativen Wichtigkeiten bleiben während des ganzen Beobachtungszeitraumes konstant. Sie sind dann im allgemeinen durch konstante Zahlenwerte repräsentiert, die reziprok zu der gegebenen Wichtigkeit sind.
- b. Die Prioritätszahlen und damit die relativen Wichtigkeiten bleiben solange konstant, bis eine explizite (d.h. programmierte) Prioritätsänderung erfolgt.

- c. Die relativen Wichtigkeiten bleiben solange konstant, bis eine explizite (d.h. programmierte) Prioritätsänderung oder eine Änderung der Konkurrenzsituation erfolgt (z.B. durch Hinzukommen einer weiteren Task). Es sei darauf hingewiesen, daß die relativen Wichtigkeiten auch dann konstant bleiben können, wenn sich die Prioritätszahlen ändern (z.B.  $\overline{p_i} = \text{const}_i \cdot t$ ).
- d. Die relativen Wichtigkeiten sind Funktionen der Zeit, d.h. daß die relative Wichtigkeit der konkurrierenden Tasks von einer zeitabhängigen Größe bestimmt wird.

Man könnte die Punkte b bis d unter dem Begriff "dynamische Prioritäten" zusammenfassen; diese Bezeichnung ist jedoch bereits für bestimmte Implementationskonzepte gebräuchlich und soll daher auch hier nur in diesem Sinne verwendet werden.

Die Strategie "FP" ("Feste Prioritäten") deckt die Gruppe a ab. Sie wird in Kapitel 3.1.1. behandelt. Die im allgemeinen Sprachgebrauch unter "Dynamischen Prioritäten" verstandene Implementations-technik ist Vertreter der Gruppe b (siehe Kap. 3.1.2.).

Die Strategien "FIFO" ("First-in-first-out", siehe Kap. 3.1.3.), "KR" ("Kürzeste Restzeit", siehe Kap. 3.1.4.) sowie das Konzept der "Kritischen Prioritäten" (siehe Kap. 3.1.8.) fallen unter die Gruppe c.

Die Strategie "KS" ("Kürzester Spielraum", siehe Kap. 3.1.5.) sowie Zeitscheibentechniken sind Vertreter der Gruppe d.

### 3.1.1. Feste Prioritäten ("FP")

Die Verwendung von konstanten Prioritätszahlen als Grundlage für die Zuteilung von Betriebsmitteln ist heute in der Prozeßrechen-technik am weitesten verbreitet. Es kommen unterschiedliche Konzepte zur Anwendung, die sich in der Implementationstechnik und hinsichtlich ihrer Flexibilität unterscheiden. Die zugrundeliegende Strategie ist jedoch immer die gleiche: Jeder Task wird eine Prioritätszahl zugeordnet. Tritt eine Konkurrenzsituation ein, so erfolgt eine Betriebsmittelzuteilung jeweils an die Task mit der höchsten Priorität entspr. der kleinsten Prioritätszahl. Diese Methode ist einfach zu implementieren und anzuwenden, bringt aber schwerwiegende Nach-

teile mit sich:

Will man ein einigermaßen gutes Zeitverhalten des Gesamtsystems erzielen, so empfiehlt es sich, jenen Tasks die höheren Prioritäten zuzuteilen, denen die kürzeren Antwortzeiten zur Verfügung stehen. In [SER72] wird gezeigt, daß selbst bei einer derartigen Auswahl der Prioritäten die Auslastung der Rechenanlage bei einer größeren Anzahl von konkurrierenden Tasks i.a. nicht größer als etwa 70% sein darf, um die Einhaltung aller Zeitschranken gewährleisten zu können. Dabei gilt als Voraussetzung, daß es sich um zyklische Tasks handelt, die nur um einen Prozessor konkurrieren.

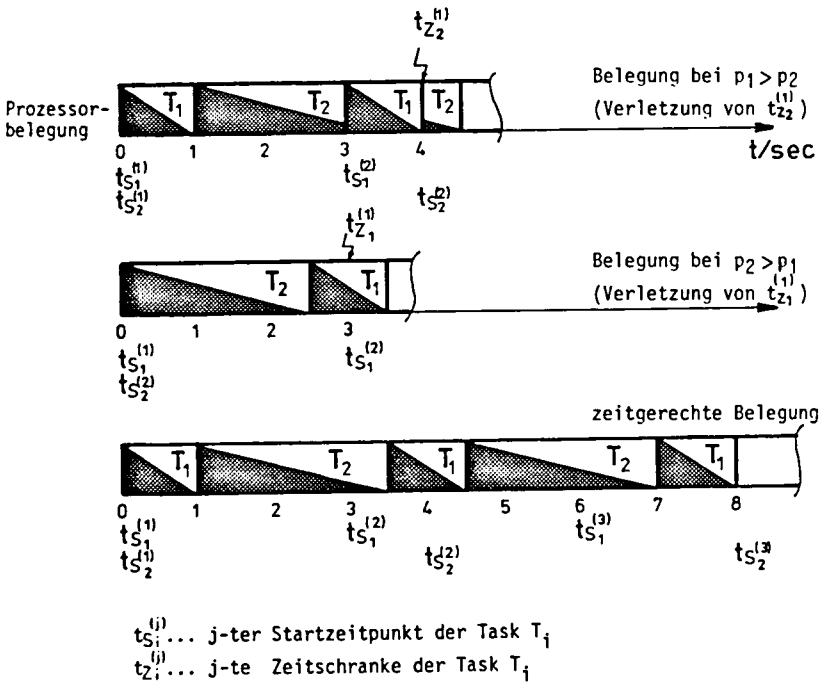
Mit einem einfachen Beispiel zweier konkurrierender Tasks kann ein Fall gezeigt werden, bei dem es keine mögliche Prioritätsbeziehung  $p_1 \neq p_2$  gibt, die befriedigende Ergebnisse liefert, obwohl grundsätzlich eine zeitgerechte Verarbeitung garantiert werden könnte:

Task $T_1$ :	Zyklus: 3 Sekunden	Laufzeit: 1 Sekunde
Task $T_2$ :	Zyklus: 4 Sekunden	Laufzeit: 2.5 Sekunden

Unter "zeitgerecht" sei hier verstanden, daß beide Tasks mit ihrer Abarbeitung jeweils innerhalb eines Zyklus' fertig werden, d.h. die Zeitschranke  $t_{zi}^{(j)}$  ist gleich dem jeweils nächsten Startzeitpunkt  $t_{si}^{(j+1)}$ . Im konkreten Fall muß also die Task  $T_1$  jeweils in den Intervallen  $[\emptyset, 3]$ ,  $[3, 6]$ ,  $[6, 9]$  .... und die Task  $T_2$  in den Intervallen  $[\emptyset, 4]$ ,  $[4, 8]$ ,  $[8, 12]$  ... vollständig abgewickelt werden.

In Bild 3.1 sind die Verhältnisse dargestellt, wie sie sich bei Anwendung der Strategie "FP" ergeben, sowie eine mögliche zeitgerechte Prozessorbelegung. Es ist interessant festzustellen, daß sich diese zeitgerechte Belegung sogar bei der Strategie "FIFO" ergeben würde, wenn man annimmt, daß die Task  $T_1$  zum ersten Mal um ein (beliebig) kleines Zeitintervall vor der Task  $T_2$  gestartet wird.





Anm.: Die Höhe der Schraffur ist ein Maß für die  
(noch bevorstehende) Laufzeit

Bild 3.1: Versagen der Strategie "FP"

### 3.1.2. "Dynamische Prioritäten"

Bleiben die vorgegebenen Prioritäten nicht über den gesamten Beobachtungszeitraum konstant, sondern hat der Anwender die Möglichkeit, prinzipiell an jeder Stelle seines Programms Prioritätsänderungen vorzunehmen, so sind einzelne, für die Strategie "FP" verwendbare, einfachere Implementationstechniken nicht mehr brauchbar (s. Kap. 3.2.2.).

Theoretisch besteht die Möglichkeit, die Abarbeitung einer Taskmenge in jeder beliebigen Reihenfolge durch geeignete Auswahl der Prioritäten erfolgen zu lassen. Jede andere Strategie kann daher prinzipiell auf eine geeignete Anwendung dynamischer Prioritäten abgebildet werden in dem Sinne, daß sich identische Abarbeitungsfolgen ergeben. "Dynamische Prioritäten" sind daher theoretisch optimal. Der Programmierer kann jedoch in der Praxis kaum in die Lage versetzt werden, die Algorithmen, die bei anderen Strategien von der Taskverwaltung ausgeführt werden, jeweils selbst im Programm zu formulieren.

### 3.1.3. First-in-first-out ("FIFO")

Bei dieser Strategie wird in einer Konkurrenzsituation jeweils diejenige Task gegenüber anderen bevorzugt, die das betreffende Betriebsmittel zum frühesten Zeitpunkt angefordert hat.

Da nun - bei einer gegebenen Konkurrenz um ein Betriebsmittel - die Reihenfolge der Tasks in der betreffenden Warteschlange unverändert bleibt, sich aber bei späteren Konkurrenzsituationen (z.B. um andere Betriebsmittel) völlig anders darstellt, ist eine Einstufung der Strategie "FIFO" gemäß Kapitel 3.1. Punkt c berechtigt.

Die Strategie "FIFO" findet teilweise im Stapelbetrieb Anwendung, wird aber auch bei Prozeßrechnern für die Vergabe von Ein/Ausgabekanälen eingesetzt [SIE69]. Die Vergabe des Prozesses erfolgt jedoch z.B. nach festen Prioritäten. Der Grund für die Kombination dieser beiden Methoden liegt wohl in der Überlegung, daß dadurch in bestimmten Fällen eine gegenseitige Blockierung von relativ unwichtigen Tasks mit viel Ein/Ausgabe-Aktivitäten vermieden werden kann.

Die Nachteile liegen aber auf der Hand: jede noch so gut überlegte Verteilung von Prioritäten wird damit bei häufiger Benutzung von E/A-Kanälen unwirksam.

Eine bessere Lösung ist wohl eine Aufteilung der Tasks nach 'Vordergrund'- und 'Hintergrund'-Aufgaben, wobei dann unterschiedliche Strategien unter der Maßgabe Anwendung finden können, daß Hintergrundtasks erst dann Betriebsmittel zugewiesen bekommen, wenn keine Anforderungen von Vordergrundtasks vorliegen.

Eine ausschließliche Verwendung der Strategie "FIFO" kann sich unter Umständen dann als vorteilhaft erweisen, wenn die folgende Bedingung gilt: Die Summe der Gesamtlaufzeiten aller beteiligten Tasks darf nicht größer als die kleinste maximal zulässige Antwortzeit sein. Dann ist eine zeitgerechte Verarbeitung garantiert und die Vorteile der Strategie "FIFO" kommen voll zum Tragen: extrem einfache Implementierbarkeit und geringer Laufzeitaufwand dadurch, daß keine Verdrängungen stattfinden.

Falls diese Bedingung nicht gilt, ist die Strategie "FIFO" der Strategie "FP" im allgemeinen unterlegen (trotz des Beispiels in 3.1.1.). Dies sei an einer einfachen Konstellation in Bild 3.2 demonstriert, wobei unter "zeitgerecht" wieder die jeweils vollständige Abarbeitung der Tasks innerhalb der zugehörigen Zyklen verstanden sei.

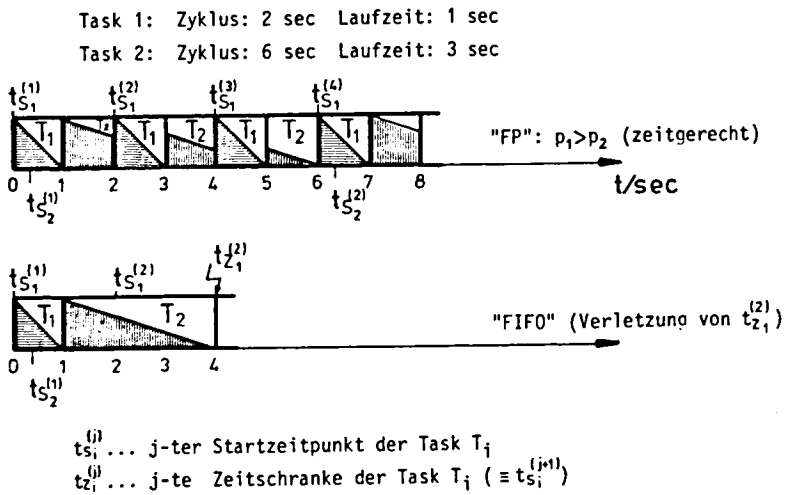


Bild 3.2: Versagen der Strategie "FIFO"

### 3.1.4. Strategie "nach der kürzesten Restzeit" ("KR")

Diese Strategie - in der Literatur auch als "next-deadline"-Strategie bezeichnet - ordnet Tasks in Warteschlangen nach aufsteigenden Restzeiten ( $r_i(t)$ ) ein. Da die Restzeit  $r_i(t)$  gleich ist der Differenz von Zeitschranke und aktueller Zeit ( $r_i(t) = t_{zi} - t$ ), kann die Zeitschranke einer Task als direkt proportional zu einer (fiktiven) Prioritätszahl betrachtet werden.

In [SER72, LIU73, HEN75, EIC75] wurde diese Strategie bereits ausführlich diskutiert und es wurde bewiesen, daß sie unter bestimmten Voraussetzungen eine optimale Verarbeitung einer Taskmenge in dem Sinne gewährleisten kann, daß die Zeitbedingungen dann eingehalten werden, wenn dies überhaupt theoretisch möglich ist. Dabei wird die Anzahl der Verdrängungen gering gehalten.

Im Unterschied zur Strategie "FP", bei der die relative Wichtigkeit aller im System befindlichen Tasks zueinander konstant bleibt, wächst bei "KR" die Wichtigkeit von bereits gestarteten Tasks im Vergleich zu der von (noch) nicht gestarteten Tasks monoton mit der Zeit. Nur die relative Wichtigkeit aller gestarteten Tasks zueinander bleibt unverändert. Dies bewirkt, daß Verdrängungen nur bei einer Änderung der Konkurrenzsituation - z.B. durch Start oder Fortsetzung einer Task - stattfinden können.

Weitere Vorteile dieser Strategie sind der Umstand, daß die Laufzeiten nicht bekannt sein müssen (im Gegensatz zu der in Kap.3.1.5. beschriebenen Strategie "KS") und die einfache Implementierbarkeit: Es ist prinzipiell kein höherer Aufwand als bei der Realisierung "dynamischer Prioritäten" erforderlich.

Im folgenden werden einige Nachteile erwähnt, sowie die einschränkenden Voraussetzungen, unter denen das erwähnte optimale Verhalten der Strategie "KR" gilt:

Bei einer stark ausgelasteten Rechananlage kann es vorkommen, daß zeitweilig Zeitschranken überschritten werden, ohne daß dies unbedingt katastrophale Folgen haben muß. Dabei kann der folgende unerwünschte Fall eintreten: Eine relativ unwichtige Task, bei der die Einhaltung der Zeitbedingungen keine sehr große Rolle spielt, wird durch den Umstand, daß ihr praktisch nie Betriebsmittel zugeteilt wurden, mit der Zeit so wichtig, daß sie allen anderen vorgezogen

wird - auch solchen, die "eigentlich" wichtiger sind.

Andererseits kann es notwendig sein, in bestimmten Notsituationen Tasks mit absoluter Priorität ablaufen zu lassen, ohne Rücksicht darauf, daß Zeitschranken anderer Tasks dadurch verletzt werden könnten.

Man kann also sagen, daß ein Einsatz von "KR" nur dann ohne Einschränkung empfohlen werden kann, wenn

- . die Anlage nur selten oder gar nie überlastet ist,
- . die Zeitschranken genau bekannt sind, und
- . die Einhaltung der Zeitbedingungen für alle Tasks annähernd "gleich wichtig" ist.

Andernfalls ergibt sich unter Umständen die Notwendigkeit einer Aufteilung der Tasks in drei Gruppen:

- . Tasks mit extrem hoher Wichtigkeit
- . "normale" Tasks
- . Hintergrundtasks

Die erste Gruppe müßte dann über feste Prioritätszahlen abgewickelt werden, die zweite Gruppe über "KR" und die dritte Gruppe wieder über feste Prioritäten oder beispielsweise "FIFO". Dadurch ergäbe sich aber sofort eine Erhöhung des Implementations- und Laufzeit-aufwandes - z.B. durch Verdreifachung der notwendigen Warteschlangen bzw. der Einkettungsmechanismen.

Die einschränkenden Voraussetzungen, unter denen der Beweis der optimalen Abarbeitung unter "KR" gilt, sind die folgenden:

- . es handelt sich um eine Einprozessoranlage
- . es erfolgt keine "langsame" Ein/Ausgabe über Kanäle
- . die Restzeit ist das einzige Kriterium für die Prozessorzuteilung an gestartete Tasks (damit darf es beispielsweise keine "kritischen Abschnitte" geben)

Durch einfache Beispiele soll gezeigt werden, daß die Strategie "KR" nicht optimal ist, wenn diese Voraussetzungen nicht gegeben sind:

# ZWEIPROZESSORANLAGE

Annahme: Drei konkurrierende Tasks  $T_1, T_2, T_3$

$$l_1(t_0)=l_2(t_0)=l_3(t_0)=1 \quad l_i \dots \text{Laufzeit der Task } T_i$$

$$r_1(t_0)=1.51, r_2(t_0)=1.61, r_3(t_0)=1.71$$

$$r_i \dots \text{Restzeit der Task } T_i$$

Bild 3.3 zeigt, daß die Strategie "KR" zu einer Verletzung der Zeitschranke  $t_{z3}$  führt, daß es aber sehr wohl eine Belegung gibt, bei der alle Zeitschranken eingehalten werden.

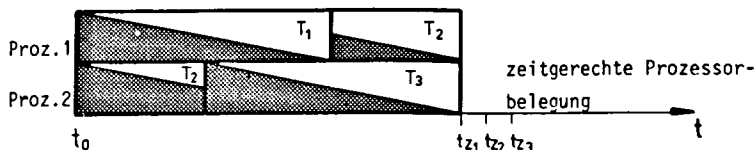
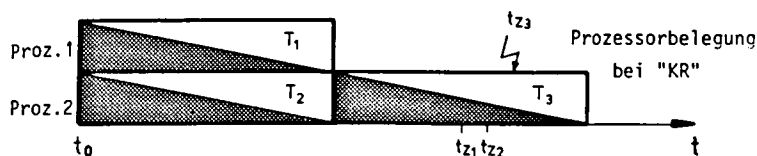


Bild 3.3: Versagen der Strategie "KR" bei Zweiprozessoranlagen

## TASKS MIT "LANGSAMER" EIN/AUSGABE

Zum Zeitpunkt  $t_0$  seien die folgenden bevorstehenden Anforderungen zweier konkurrierender Tasks gegeben:

$T_1$  benötigt für eine Zeiteinheit den Prozessor und anschließend für sechs Zeiteinheiten ein Kanalwerk.

$T_2$  benötigt für sechs Zeiteinheiten den Prozessor.

Restzeiten:  $r_1(t_0)=11, r_2(t_0)=10$

Bild 3.4 zeigt das Versagen der Strategie "KR" bei dieser Konstellation.

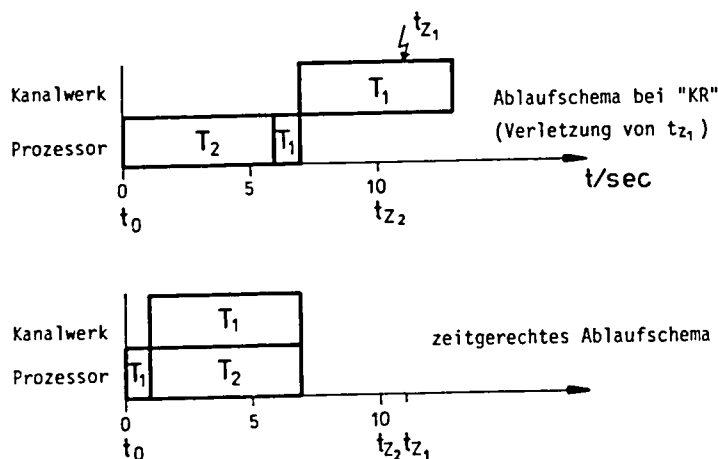


Bild 3.4: Versagen der Strategie "KR" bei "langsamer" Ein/Ausgabe

### "KRITISCHE ABSCHNITTE"

Angenommen seien 3 Tasks, die während ihres Ablaufes je einen kritischen Abschnitt bezüglich einer gemeinsamen Variablen betreten. Bild 3.5 zeigt das angenommene Anforderungsschema, die Verletzung der Zeitschranke  $t_{z2}$  bei Anwendung der Strategie "KR" sowie eine zeitgerechte Prozessorbelegung.

Es sei hier darauf hingewiesen, daß mit diesen Beispielen nicht unterstellt werden soll, daß eine andere bekannte Strategie unter den erschwerenden Voraussetzungen bessere Ergebnisse liefert als die Strategie "KR". Es wird nur unterstrichen, daß der Beweis der optimalen Verarbeitung unter "KR" in diesen Fällen nicht geführt werden kann.

Stellt man nun die Frage, ob trotz dieser Aussage die Verwendung von "KR" auch unter den genannten Bedingungen zu empfehlen ist, so kann dies mit Einschränkungen bejaht werden: Beispielsweise kann bei Ein/Ausgabeoperationen, deren Zeitdauer im Vergleich zu

den für die Strategie relevanten Zeitintervallen (Laufzeiten, Restzeiten) klein ist, der Anteil an Kanalbelegungszeiten sicher vernachlässigt werden. Außerdem weiß man aus der Warteschlangentheorie, daß die Strategie "KR" bei der Konkurrenz um "nicht preemptive" Betriebsmittel die "mittlere Verspätung" der Aufträge minimiert, was zum Teil auch für Realzeitanwendungen von Interesse sein kann [HOF74] .

Beim Auftreten kritischer Abschnitte kann bei zusätzlicher Anwendung des in Kap.3.1.8. vorgeschlagenen Konzepts der "kritischen Prioritäten" eine Verbesserung erzielt werden.

Anforderungen:

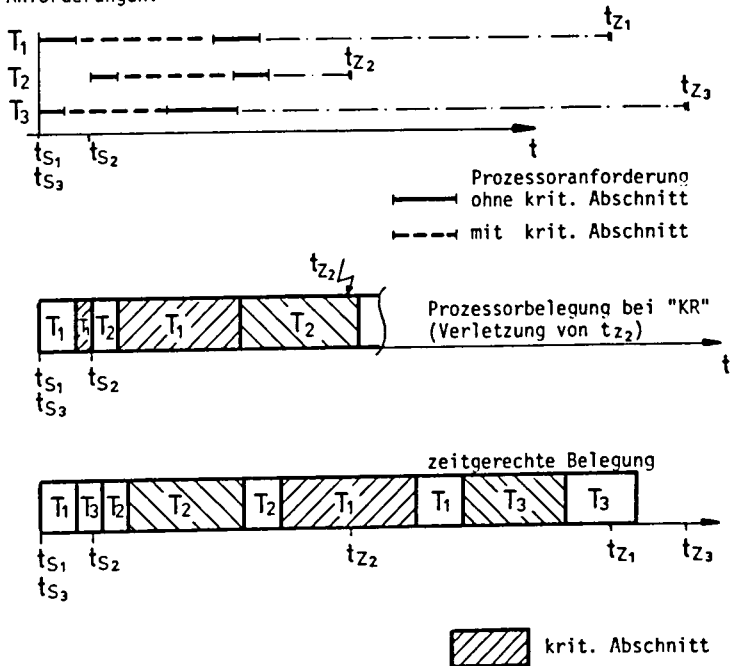


Bild 3.5: Versagen der Strategie "KR" bei "kritischen Abschnitten"



### 3.1.5. Strategie "nach dem kleinsten Spielraum" ("KS")

Die Strategie nach dem kleinsten Spielraum (oder "slack-time"-Strategie) beruht darauf, daß jene Tasks bevorzugt bedient werden, bei denen der Wert

$$s(t) = r(t) - l(t)$$

(Spielraum = Restzeit - Laufzeit)

am kleinsten ist.

In [HEN75] wird bewiesen, daß diese Strategie sowohl bei Ein- als auch Mehrprozessoranlagen eine zeitgerechte Verarbeitung von Taskmengen gewährleistet, deren Tasks an jeder Stelle wechselseitig verdrängbar sind, und die ebenfalls zwei der bereits bei der Beschreibung von "KR" erwähnten einschränkenden Voraussetzungen genügen: Es dürfen keine "langsamen" Ein/Ausgaben und keine kritischen Abschnitte auftreten.

Sie ist demnach bei Einprozessoranlagen der Strategie "KR" theoretisch gleichwertig und bei Mehrprozessoranlagen überlegen. Ihre hauptsächlichen Nachteile sind die schwierige Implementierbarkeit und der beträchtliche Laufzeitaufwand.

Im Gegensatz zu "KR", wo die Prioritätsbeziehungen der Tasks solange konstant bleiben, als sich die Konkurrenzsituation nicht durch neu hinzukommende Tasks ändert, und demnach keine Verdrängungen erforderlich sind, ist dies bei "KS" nicht der Fall:

Bei Tasks, deren Spielraum größer ist, reduziert sich dieser mit der Zeit stärker als bei anderen, was eine Tendenz zur Angleichung der Spielräume zur Folge hat. Dieses Phänomen beruht darauf, daß bei Tasks, denen auf Grund geringerer Wichtigkeit kein Prozessor zugeteilt ist, im Ausdruck  $r(t)-l(t)$  der Wert  $l(t)$  konstant bleibt, wogegen er sonst monoton fällt. Würden bei einer Anzahl von Tasks mit gleichem Spielraum nur ein Teil (bzw. eine) für eine Prozessorzuteilung ausgewählt, so hätte dies sofort zur Folge, daß der Spielraum der wartenden Tasks kleiner würde als der der ausgewählten Task(s). Damit wäre aber die Bedingung für die Strategie verletzt. Es müssen daher Vorkehrungen getroffen werden, damit allen Tasks mit dem gleichen, kleinsten Spielraum in jedem Zeitintervall  $\Delta t$  ein gleicher Anteil an Prozessorzeit zugeteilt wird. Die Konsequenz daraus sind Verdrängungen in möglichst kleinen Zeitabstän-

den, um diese Forderung wenigstens näherungsweise zu erfüllen. Da überdies die Notwendigkeit besteht, an u.U. sehr vielen Stellen des Programmablaufes die Prioritätsordnung neu zu bestimmen, und da hierzu die Laufzeiten in Abhängigkeit von der Zeit bekannt sein müssen, ergeben sich große implementationstechnische Schwierigkeiten und ein für die Praxis kaum zu vertretender Laufzeitaufwand.

### 3.1.6. Der Begriff der "Restzeitreduktion"

Die theoretische Gleichwertigkeit der Strategien "KR" und "KS" bei Einprozessoranlagen läßt sofort die Frage aufkommen, ob ein Kriterium gefunden werden kann, das eine Klasse von "zeitgerechten" Strategien definiert, wobei "KR" und "KS" Vertreter dieser Klasse sind. Dieses Kriterium lautet wie folgt:

Eine Strategie ist an einer Einprozessoranlage bei ausschließlicher Konkurrenz um einen Prozessor dann zeitgerecht, wenn sie während jedes Zeitintervalls  $\Delta t$  einer Tasks den Prozessor zuteilt, bei der die Summe aus Spielraum und  $\Delta t$  kleiner oder gleich ist als die kürzeste Restzeit aller konkurrierenden Tasks.

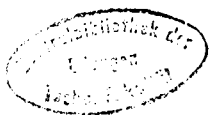
Der Beweis für diesen Satz ist in Anhang I erbracht.

Man kann sich nun sehr einfach davon überzeugen, daß sowohl "KR" als auch "KS" diese Bedingung erfüllen:

Die Strategie "KR" teilt immer der Task mit der kürzesten Restzeit den Prozessor zu. Der Spielraum jeder Task ist aber kleiner als ihre Restzeit ( $s_i(t) = r_i(t) - l_i(t)$ ), womit das Kriterium für  $\Delta t \leq l_i(t)$  erfüllt ist.

Die Strategie "KS" teilt immer der Task mit dem kleinsten Spielraum den Prozessor zu. Es kann nun aber keine Task geben, deren Restzeit kleiner ist als dieser Spielraum, da ja sonst deren Spielraum noch kleiner sein müßte.

Aus dem genannten Kriterium läßt sich eine interessante Schlußfolgerung ableiten: Man kann die Restzeit jeder beliebigen Task einer zeitgerecht verarbeitbaren Taskmenge um einen beliebigen Anteil ihrer noch bevorstehenden Laufzeit reduzieren und dann solange nach "KR" vorgehen, bis (spätestens) die betreffende Task nur mehr diesen Laufzeitanteil abzuwickeln hat, und erhält so wieder eine zeitgerecht verarbeitbare Taskmenge (siehe Anhang I).



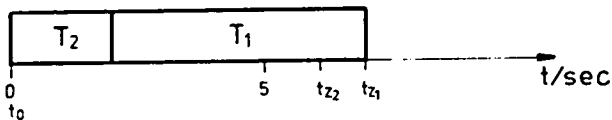
Dies sei an einem einfachen Beispiel demonstriert:

Gegeben sei eine Konkurrenzsituation zweier Tasks. Die Laufzeiten und Restzeiten seien wie folgt angenommen:

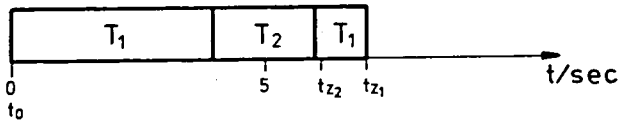
$$l_1(t_0) = 5, l_2(t_0) = 2$$

$$r_2(t_0) = 7, r_2(t_0) = 6.1$$

Die Strategie "KR" würde der Task  $T_2$  zuerst den Prozessor zuteilen, wodurch sich die folgende Prozessorbelegung ergeben würde:



Bei einer Restzeitreduktion um eine Zeiteinheit für die Task  $T_1$  würde der Prozessor zuerst für vier Zeiteinheiten der Task  $T_1$  zugeweiht. Spätestens dann muß sie unterbrochen werden, da nun nur mehr die eine Einheit an Laufzeit bevorsteht, um die die Restzeit reduziert wurde. Dabei ergäbe sich die folgende Prozessorbelegung:



Man sieht, daß auch hier die Zeitbedingungen eingehalten werden.

Auf den ersten Blick scheinen solche Restzeitreduktionen nicht sinnvoll zu sein, wo sich doch damit die Situation gegenüber der Strategie "KR" dadurch verschlechtert, daß zumindest die berücksichtigten Anteile der Tasklaufzeiten bekannt sein müssen und daß sich u.U. eine größere Anzahl von Verdrängungen ergibt. Im folgenden Abschnitt wird gezeigt, unter welchen Voraussetzungen diese Vorgangsweise trotzdem positive Auswirkungen haben kann.

### 3.1.7. Berücksichtigung "langsamer" Ein/Ausgabe

Bestehen konkurrierende Tasks nicht nur ausschließlich aus (preemptiven) Prozessorbelegungssequenzen, sondern werden z.B. auch Datentransfers von/zu "langsamen" E/A-Geräten ausgeführt, so liefern die Strategien "KR und "KS" nicht mehr in jedem Fall zeitgerechte Belegungsfolgen.

Man kann im Gegenteil sogar beweisen, daß es unter diesen Bedingungen keine optimale Strategie geben kann, sofern eine Strategie dann als optimal zu verstehen ist, wenn sie ohne Kenntnis zukünftiger Ereignisse eine zeitgerechte Betriebsmittelbelegungsfolge dann liefert, wenn es eine solche gibt.

Ein Beispiel ist in Bild 3.6 angeführt. Eine Strategie, die die angegebene zeitgerechte Belegung liefern sollte, müßte zum Zeitpunkt  $t=0$  davon Kenntnis haben, daß die Task  $T_2$  zum Zeitpunkt  $t_{S2}=1$  gestartet wird und dürfte dementsprechend der Task  $T_1$  das Kanalwerk nicht zuteilen. Dies kann natürlich keinesfalls vorausgesetzt werden, wenn der Start von Task  $T_2$  beispielsweise durch ein stochastisches Ereignis erfolgt.

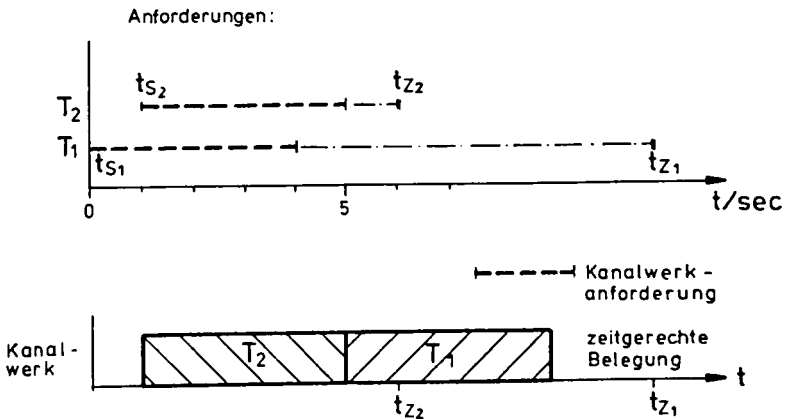


Bild 3.6: Konkurrenz zweier Tasks um ein Kanalwerk

Selbstverständlich ist nun das in Kap.3.1.6. erwähnte Kriterium für zeitgerechte Abarbeitung auch nicht mehr gültig. Die praktische Erfahrung zeigt jedoch, daß die Strategie "KR" im allgemeinen trotzdem noch bessere Ergebnisse liefert als z.B. die Strategie "FP". Nun erhebt sich die Frage, ob durch irgendwelche nicht zu komplizierte Maßnahmen eine weitere Verbesserung zu erzielen ist. Ein möglicher Weg wird durch die "Restzeitreduktion" aufgezeigt: Falls bei einem gegebenem Programmsystem die Abläufe der einzelnen Tasks insoweit bekannt sind, als die Folgen von Prozessorbelegungsphasen und langsamen Ein/Ausgaben festliegen (z.B. bei einem System von DDC-Tasks: Prozessorbelegung-Analogeingabe-Prozessorbelegung-Digitalausgabe-Prozessorbelegung) und außerdem die Kanalzeiten zumindest näherungsweise bekannt sind, so gehe man wie folgt vor:

Man reduziere die Restzeit jeder Task um nur die Kanalzeiten, die (außer einer möglicherweise bereits laufenden) noch bevorstehen, und wiederhole diesen Schritt nach jeder Kanalbelegung.

Die zugrundeliegende Überlegung ist die, daß es von Vorteil ist, Tasks mit viel Ein/Ausgabe gegenüber anderen etwas zu bevorzugen, da während deren Kanalbelegungen der Prozessor ohnedies anderen Tasks zur Verfügung steht. Durch das Beispiel in Bild 3.7 sei dies illustriert: Die Strategie "KR" führt zu einer Verletzung der Zeitschranke  $t_{22}$ , wogegen durch die empfohlene Variante der "Restzeitreduktion" ( $t_{22}' = t_{22} - 4 = 8$ ) und der dementsprechend bevorzugten Behandlung von  $T_2$  eine Einhaltung der Zeitschranken gewährleistet ist. Dies entspricht in etwa der heuristischen Lösung in Bild 3.4.

Es muß an dieser Stelle gesagt werden, daß sehr wohl Fälle konstruiert werden können, für die eine andere Vorgangsweise bessere Ergebnisse liefert, doch dürfte für den Großteil der möglichen Konstellationen eine Überlegenheit der empfohlenen Strategie gegeben sein. Dieser Standpunkt wird durch die Beschreibung praktischer Versuche in Kapitel 5 erhärtet.

Bei dem Sonderfall einer Taskmenge, die zum Zeitpunkt  $t_0$  einerseits aus Tasks besteht, die nur den Prozessor anfordern, andererseits aus Tasks, die die Folge Prozessor-Kanalwerk anfordern, liefert die Strategie "KR" mit "Restzeitreduktion" - unter Verwendung der Kanalzeit als Reduktion - ebenso gute bzw. bessere Ergebnisse als die Strategie "KR". Den Beweis findet man im Anhang I.

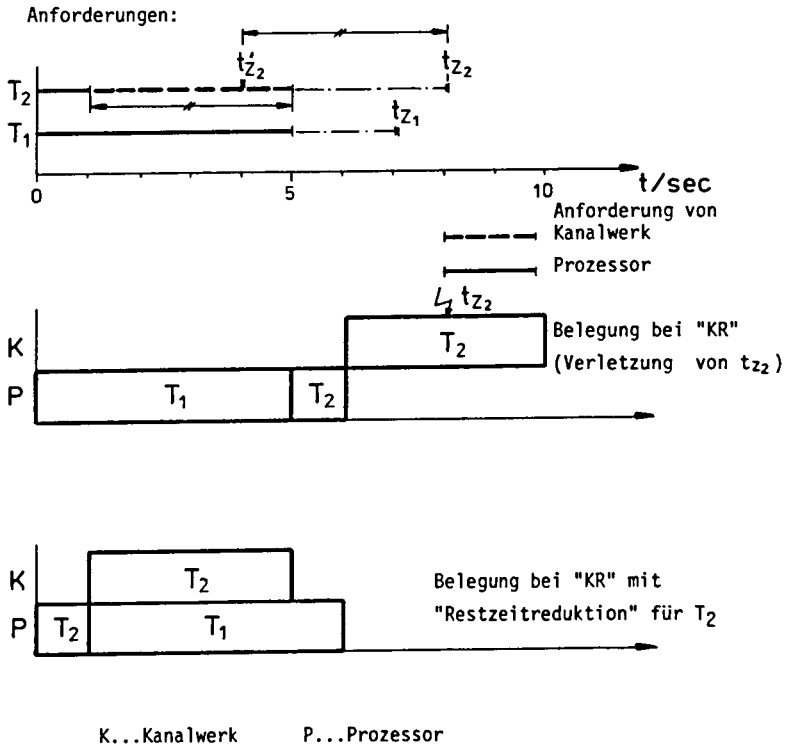


Bild 3.7: Vorteilhafte Anwendung der "Restzeitreduktion"

### 3.1.8. Berücksichtigung von "kritischen Abschnitten"

Ähnlich wie beim Auftreten von "langsamen" E/A-Operationen kann auch bei der Verwendung von kritischen Abschnitten gezeigt werden, daß die Strategien "KR" und "KS" nicht mehr in jedem Fall optimale Ergebnisse liefern. Die im Folgenden beschriebene Situation hat bei allen bisher erwähnten Strategien nachteilige Konsequenzen:

Es seien drei Tasks  $T_1, T_2, T_3$  angenommen mit den Prioritäten  $p_1 > p_2 > p_3$ .  $T_3$  befinde sich in einem kritischen Abschnitt und  $T_1$  sei deshalb am Eintreten in einen korrespondierenden kritischen Abschnitt gehindert.  $T_2$  belege für relativ lange Zeit den Prozessor (ohne kritischen Abschnitt). Wegen  $p_2 > p_3$  wird  $T_3$  solange zurückgestellt, bis  $T_2$  "freiwillig" den Prozessor abgibt. Damit wird aber auch die wichtige Task  $T_1$  mindestens ebenso lange am Weiterlaufen gehindert. Aus diesem Grund liegt die folgende Vorgangsweise nahe:

Befindet sich eine Task geringer Wichtigkeit (bzw. größerem  $r(t)$ ) in einem kritischen Abschnitt, und verhindert sie dadurch die Fortsetzung einer Task größerer Wichtigkeit, so wird sie für die Dauer der Bearbeitung des kritischen Abschnitts als ebenso wichtig betrachtet wie die blockierte Task. Dadurch werden Laufhindernisse für wichtige Tasks beschleunigt beseitigt.

Die Priorität, die eine Task bei dieser Methode innerhalb eines kritischen Abschnitts erhält, wird im Folgenden als "kritische Priorität" bezeichnet.

Dieses Konzept kann sowohl bei der Verwendung üblicher Prioritätszahlen ("FP" oder "dynamische Prioritäten") als auch bei der Strategie "KR" zusätzlich eingesetzt werden. Der dafür notwendige Implementations- und Laufzeitaufwand ist allerdings nicht unerheblich (siehe Kapitel 4).

### 3.2. Implementations- und Anwendungsaspekte

In diesem Abschnitt wird kurz umrissen, welche Anforderungen an eine Implementation verschiedener, in dieser Arbeit erwähnter Zuteilungsstrategien für Einprozessoranlagen gestellt werden, und welche Vor- und Nachteile sich bei ihrer Anwendung ergeben.

#### 3.2.1. Feste Prioritäten ("FP")

Diese Strategie ist neben "FIFO" wohl am einfachsten zu implementieren. In den sogenannten "Taskkontrollblöcken" (Taskverwaltungsblöcken, Prozeßkontrollblöcken) - im Folgenden mit "TKB" abgekürzt - die die für die Verwaltung der Tasks notwendigen Informationen enthalten, kann die Priorität als eine Konstante eingetragen sein. Die

Organisation der Warteschlangen kann je nach Komfortstufe auf die folgenden Arten realisiert sein:

- a. Unbegrenzter Vorrat an Prioritätsnummern und Möglichkeit, verschiedene Tasks mit gleichen Prioritäten zu belegen.

Hierbei empfiehlt es sich, die Warteschlangen als "Vorwärts-Rückwärts-Verkettung" der betroffenen Taskkontrollblöcke auszuführen (siehe Bild 3.8).

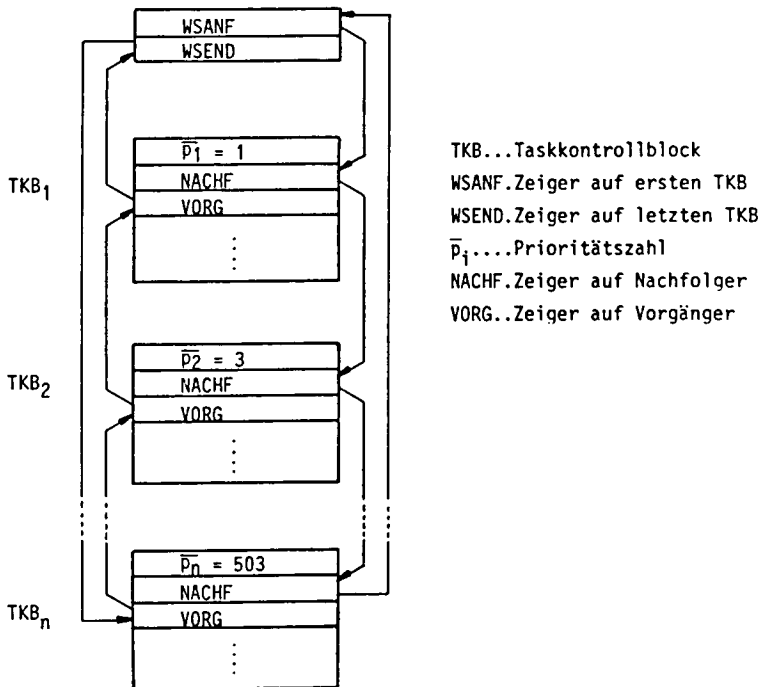


Bild 3.8: Warteschlange mit "Vorwärts-Rückwärts-Verkettung"



Dieses Verkettungsverfahren ermöglicht eine besonders schnelle Auskettung eines Taskkontrollblocks. Außerdem gewährleistet die Art der Verkettung mit den "Kopfzellen" WSANF und WSEND einen Mechanismus des Ein- und Austragens, der unabhängig davon ist, ob sich der betreffende TKB am Anfang, am Ende oder in der Mitte der Warteschlange befindet.

Bei einer Verkettung in nur einer Richtung müßte beim Ausketten ein Suchvorgang ablaufen, um den "Vorgänger" (bzw. "Nachfolger") zu bestimmen.

Die Einkettung kann bei diesem Verfahren nur über einen Suchvorgang erfolgen. Es muß ja über die Priorität die richtige Stelle gefunden werden, an der der betreffende TKB einzuordnen ist. Dadurch kann sich ein nicht unerheblicher Laufzeitaufwand ergeben, wenn die Anzahl der gleichzeitig um ein Betriebsmittel konkurrierenden Tasks groß ist.

- a. Begrenzter Vorrat an Prioritäten und Möglichkeit, verschiedene Tasks mit gleichen Prioritäten zu belegen.

Es empfiehlt sich, die Warteschlange in gleicher Form wie in Bild 3.8 auszuführen. Parallel dazu kann aber eine Tabelle geführt werden, deren Elementanzahl gleich ist dem Vorrat an Prioritätsnummern. Die Elemente stellen dann jeweils Zeiger auf den ersten TKB mit der betreffenden Priorität dar. Dadurch wird der Suchaufwand für den Einkettungsvorgang erheblich reduziert, wenn nicht eliminiert [EIC75].

Ein Nachteil dieser Methode besteht darin, daß diese Tabelle im Prinzip für jede Warteschlange - also jedes Betriebsmittel getrennt geführt werden muß, wodurch sich u.U. ein großer Speicherbedarf ergibt. Man kann nun davon ausgehen, daß nur bei der Konkurrenz um den Prozessor die schnelle Einkettung eines TKB von Wichtigkeit ist, da vielfach die Warteschlangen der anderen Betriebsmittel im Durchschnitt relativ kurz sind. Man käme dann mit einer Tabelle aus, müßte aber verschiedene Ein- und Auskettungsmechanismen in Kauf nehmen.

c. Begrenzter Vorrat an Prioritäten und Verbot, verschiedene Tasks mit gleichen Prioritäten zu belegen.

Unter diesen einschränkenden Bedingungen kann ein anderes Verfahren der Warteschlangendarstellung angewandt werden, bei dem die Einkettung besonders schnell vonstatten geht:

Jede Warteschlange wird als eine Bitkette der Länge  $n$  ausgeführt, wobei  $n$  die Anzahl der verfügbaren Prioritätsnummern ist. Ein gesetztes Bit bedeutet, daß die Task mit der betreffenden Priorität sich in der Warteschlange befindet. Diese Methode ist besonders platzsparend und effektiv, wenn die Anzahl der Prioritäten auf die Wortlänge der betreffenden Maschine beschränkt ist [SIE69].

Allen drei erwähnten Verfahren ist gemeinsam, daß die Task, die ein Betriebsmittel gerade belegt, entweder besonders ausgezeichnet ist, oder durch den vordersten Eintrag in der Warteschlange repräsentiert ist.

Bei der Anwendung von festen Prioritäten können diese entweder beim Laden eines Programmsystems durch "job-control" vergeben, oder auf Sprachebene bei der Deklaration einer Task vorgesehen werden - wie dies z.B. auch in PEARL [PDV77] möglich ist:

MESSUNG: TASK PRIORITY 10

### 3.2.2. Dynamische Prioritäten

Dynamische Prioritäten sind dadurch gekennzeichnet, daß es möglich ist, zur Laufzeit eines Programmsystems die Prioritätseinträge in den Taskkontrollblöcken zu verändern. Damit ergibt sich die Notwendigkeit des "Umkettens" in Warteschlangen. Da dies aber durch "Ausketten" und anschließendes "Einketten nach Priorität" ausgeführt werden kann, finden auch hier die für "FP" in Kap 3.2.1. unter Punkt a und b erwähnten Implementationsverfahren Anwendung. Das unter Punkt c erwähnte, effizientere Verfahren ist nicht mehr einsetzbar, da das Verbot der Doppelbelegung von Prioritäten nicht mehr sinnvoll aufrechtzuerhalten ist.

Für die Anwendung von dynamischen Prioritäten muß dem Programmierer

ein Mittel in die Hand gegeben werden, die Prioritätsfestlegung an beliebigen Stellen im Programm vorsehen zu können. Dies geschieht am besten auf Sprachebene durch ein "UPDATE-PRIORITY"-Statement wie in EAL/S [HEW74] oder wie in PEARL [PDV77] durch das "PRIORITY"-Attribut beim "ACTIVATE"- bzw. "CONTINUE"-Statement.

### 3.2.3. First-in-first-out ("FIFO")

Bei der Strategie "FIFO" ist die Verwaltung der Warteschlangen besonders einfach, da diese nur in einer Richtung verkettet sein müssen. Einzige Bedingung dafür ist, daß eine Task nur durch sich selbst und nicht durch eine fremde Task aus der Warteschlange ausgetragen werden darf. Dann erfolgt ja ein "Einketten" immer an das Ende und ein "Ausketten" immer vom Anfang der Warteschlange.

Für die Strategie "FIFO" brauchen (bzw. können) auf Sprachebene keine Hilfsmittel zur Unterstützung des Zuteilungsmechanismus existieren.

### 3.2.4. Strategie nach der kürzesten Restzeit ("KR")

Eine effiziente Implementationsmethode für die Strategie "KR" wird in [EIC75] angegeben. Sie kann mit dem in Kapitel 3.2.1. unter Punkt b diskutierten Verfahren verglichen werden - was die schnelle Einkettung betrifft, unterscheidet sich in der sonstigen Technik aber erheblich davon.

Es besteht überdies die Möglichkeit, ein "normales" Warteschlangenverfahren (wie in Kap.3.2.1, Punkt a) anzuwenden: Dazu ist es nur notwendig, zum Zeitpunkt der Vorgabe einer Restzeit die Summe aus Restzeit und aktueller Zeit zu bilden und dies als Prioritätszahl zu betrachten (diese Summe stellt ja die Zeitschranke dar, die keine Funktion der Zeit - also eine Konstante ist).

Die einzige kleine Schwierigkeit besteht darin, daß dieser Wert wegen der wachsenden aktuellen Zeit irgendwann nicht mehr darstellbar ist - bzw. daß mit der Darstellung der Zeit im Rechner in gewissen Abständen immer wieder "von vorne" begonnen werden muß.

Dies hat zur Folge, daß der beim Einketten notwendige Prioritätsvergleich die Tatsache berücksichtigen muß, daß nicht immer höhere Prioritäten durch kleinere Zahlenwerte repräsentiert sind.

Man kann sich jedoch leicht davon überzeugen, daß bei den meisten Rechnern dieses Problem sehr einfach dadurch gelöst werden kann, daß für den Prioritätsvergleich ein Subtraktionsbefehl verwendet wird, wobei ein etwaiger Überlauf unberücksichtigt bleibt. Dann ist das Ergebnis eindeutig, solange keine Zeitschranke um einen größeren Betrag überschritten wird als die Differenz zwischen (vorzeichenbehafteter) Größtzahl und größter vorgegebbarer Restzeit.

Diese Methode wurde auch bei dem im Zusammenhang mit dieser Arbeit implementierten Modellbetriebssystem verwendet. Sie bietet den Vorteil, daß im Prinzip der gleiche Mechanismus für die Strategien "KR", "Dynamische Prioritäten" (und damit natürlich auch "FP"), sowie "FIFO" benützt werden kann. Die einzige notwendige Änderung beim Übergang von "KR" zu "FP" (bzw. "Dynamischen Prioritäten") ist eine Unterbindung des Hochzählens der erwähnten Zeitzelle durch den Taktgeber. Um zu "FIFO" zu gelangen, muß nur noch allen Tasks die gleiche Priorität zugewiesen werden (vorausgesetzt ist ein Vorgehen nach "FIFO" bei gleichen Prioritäten).

Um die Strategie "KR" sinnvoll anwenden zu können, sollten Sprachmittel vorhanden sein, die eine dynamische Vereinbarung von Restzeiten ermöglichen. Ein Vorschlag dafür wurde bereits in [EIC75] gemacht.

Die Forderung, daß eine bestimmte Aktion spätestens zu einem bestimmten Zeitpunkt erfolgen soll, ist jedoch nur ein Spezialfall der verschiedenen denkbaren Zeitbedingungen. In Kapitel 2 wurde eine Klassifizierung der in der Praxis auftretenden zeitlichen Anforderungen vorgenommen. Dabei wurde festgestellt, daß Probleme der Gruppen a ("Zeitpunkte mit Toleranzen"), c ("spätestens"), und d ("frühestens") mit zeitschrangesteuerten Strategien ("KR", "KR" mit "Restzeitreduktion") relativ gut lösbar sind (siehe Kap.2.6, Bild 2.13).

Zeitbedingungen der Gruppe d ("frühestens") stellen prinzipiell keine Schwierigkeit dar und sind mit bereits eingeführten Sprachmitteln (z.B. "SCHEDULES" in PEARL) beschreibbar.

Das in [EIC75] vorgestellte Konzept ermöglicht die Beschreibung von Zeitbedingungen der Gruppe c ("spätestens"). Ein Beispiel ist die

Angabe

WHEN ALARM ACTIVATE NOTTASK EXECTIME 0.2 SEC;

Für die Beschreibung von Zeitbedingungen der Gruppe a (Sollzeitpunkte mit Toleranzen) wird der folgende Lösungsweg vorgeschlagen: Als Zeitschranke der Task wird die obere Toleranzgrenze angegeben. Kurz vor Ausführung der relevanten Aktion wird eine "P-Operation" [1372] auf einer Semaphor-Variablen ausgeführt. Die zugehörige "V-Operation" wird über einen "Schedule" derart eingeplant, daß sie beim Erreichen der unteren Toleranzgrenze durch das Betriebssystem ausgeführt wird. Ein Beispiel soll dies verdeutlichen:

```
1 M: WHEN EREIGNIS AFTER INTERVALL-TOLERANZ RELEASE S;
2   WHEN EREIGNIS RESUME EXECTIME INTERVALL+TOLERANZ;
```

.  
.  
.

Algorithmus zur Berechnung einer Stellgröße

.  
.

```
3   REQUEST S;
4   SEND FROM STELLGRÖSSE TO STELLGLIED;
5   GOTO M;
```

Durch die Anweisung in Zeile 1 wird gewährleistet, daß die Task nach dem Eintreffen des Ereignisses frühestens nach der Zeitspanne INTERVALL-TOLERANZ die Anweisung in Zeile 3 überlaufen kann. Durch die Festlegung der Zeitschranke in Zeile 2 wird (nach Möglichkeit) ein rechtzeitiges Anstoßen der Ausgabe in Zeile 4 vorgesehen. Damit dies möglich ist, sollte die angegebene Toleranz groß sein im Vergleich zur Ausführungszeit der Anweisungen in den Zeilen 3 und 4 - sofern diese nicht bereits bei der Festlegung der unteren Toleranzgrenze berücksichtigt wurde. Je nach Auslastung der Anlage wird der Stellbefehl früher oder später innerhalb der vorgegebenen Toleranzgrenzen erfolgen.

Dieses Beispiel zeigt, daß es zu einer sinnvollen Ergänzung von

Strategien, die die Einhaltung von Zeitschranken zum Ziel haben, möglich sein sollte, die Semaphore-Operation "RELEASE" mit einem "SCHEDULE" zu verknüpfen.

### 3.2.5. Strategie nach dem kürzesten Spielraum ("KS")

Da die Strategie "KS" mit erheblichen implementationstechnischen Schwierigkeiten verbunden ist, und da auch in der Praxis im allgemeinen nicht davon ausgegangen werden kann, daß die Tasklaufzeiten bekannt sind, wird hier nicht näher darauf eingegangen.

### 3.2.6. "Restzeitreduktion"

Man kann das Konzept der Restzeitreduktion auf einer praktisch unveränderten Implementation der Strategie "KR" aufbauen. Es bleibt dann dem Programmierer überlassen, die Reduktion der Restzeiten explizit vorzusehen. Zur Wiederanhebung vorübergehend reduzierter Restzeiten ist dann allerdings ein zusätzliches Sprachmittel erforderlich.

Die Anwendung sei wieder an einem Beispiel demonstriert:  
Alle 0.5 Sekunden sei von einem (integrierenden) ADC ein Meßwert einzulesen. Die Integrationszeit betrage 20 Millisekunden.  
Danach sei über einen Algorithmus eine Stellgröße zu berechnen, die über einen (schnellen) DAC an ein Stellglied ausgegeben wird, wobei die gesamte Task spätestens nach 0.4 Sekunden abgeschlossen sein sollte.

STARTTASK: TASK

·  
·

ADCTIME = 0.02 SEC;

ALL 0.5 SEC ACTIVATE DDC EXECTIME 0.4 SEC - ADCTIME;  
/\* Restzeit um Integrationszeit des ADC reduziert \*/

·  
·

END;

DDC: TASK

DCL (MESSWERT, STELLWERT) FIXED;  
TAKE FROM ADC TOM MESSWERT;

```

INCREASE EXECTIME BY ADCTIME; /* Wiederanhebung der
                                Restzeit */

      .
      .
      .
/* Algorithmus */
      .
      .
SEND FROM STELLWERT DO DAC;
END;

```

Die Anweisung zur Erhöhung der Restzeit könnte auch dargestellt werden durch

$$\text{EXECTIME} = \text{EXECTIME} + \text{ADCTIME};$$

falls der Name EXECTIME als reservierter Systemname einen Zugriff auf die Prioritätszelle (bzw. Zeitschranke) der Task ermöglichte.

### 3.2.7. "Kritische Prioritäten"

Wie bereits erwähnt, erfordert die Realisierung des Konzepts der "Kritischen Prioritäten" einen nicht unerheblichen Implementationsaufwand. In der Beschreibung des Modellbetriebssystems, das im Rahmen dieser Arbeit erstellt wurde, ist ein Lösungsweg aufgezeigt (siehe Kap.4.3.5.).

Die dadurch mögliche Modifizierung der unterlagerten Taskverwaltungsstrategie allein rechtfertigt diesen Aufwand nicht unbedingt. Da jedoch - gleichsam als Nebenprodukt - ein Mechanismus zur Erkennung von Systemverklebungen abfällt, erhöht sich die Attraktivität des Konzepts in beträchtlichem Maße.

Neue Sprachmittel sind hier nicht erforderlich, da die möglichen Prioritätsänderungen nur implizit erfolgen.

#### 4. IMPLEMENTATION VON TASKVERWALTUNGSSTRATEGIEN

##### 4.1. Allgemeines

Im Rahmen dieser Arbeit wurde ein Modellbetriebssystem entworfen und implementiert, mit dessen Hilfe Vor- und Nachteile der folgenden Strategien bzw. Strategieerweiterungen demonstriert werden sollten:

- . Feste Prioritäten ("FP")
- . Kürzeste Restzeit ("KR")
- . Strategie "KR" mit "Restzeitreduktion"
- . "Kritische Prioritäten"

Es wurde daher besonderes Gewicht auf die Realisierung jener Programmbausteine gelegt, die die Verwaltung der Tasks (Rechenprozesse) sowie die Zuteilung der Betriebsmittel an dieselben zur Aufgabe haben. Dabei wurde jedoch aus Aufwandsgründen die Organisation von Speicherplatz ausgeklammert, obwohl nicht verkannt wurde, daß eine eingesetzte Strategie letztlich nur dann befriedigende Ergebnisse liefern kann, wenn sie eine effiziente Lösung dieses Problems zumindest nicht ausschließt.

Für jene Aufgaben, die bei einem praxisnahen Echtzeitbetriebssystem ebenfalls von Bedeutung sind, die aber auf das Zeitverhalten (bzw. auf die Strategien zur Steuerung des Zeitverhaltens) kaum Einfluß haben, wurden nur Primitivlösungen implementiert. In diesem Zusammenhang sind zu nennen:

- . die Kommunikation zwischen Operateur und Rechner
- . die Behandlung von Laufzeitfehlern
- . das Laden eines Programmsystems etc.

##### 4.1.1. Fähigkeiten des Modellbetriebssystems

Um eine möglichst einfache Programmierung von Anwenderprogrammen zu ermöglichen, deren Verhalten bei der Verwendung der erwähnten Strategien geprüft werden sollte, wurde davon ausgegangen, daß dafür die Programmiersprache PEARL verwendet wird. Die Ablauforganisation wurde daher derart implementiert, daß eine ausreichende Teilmenge



der in PEARL vorgesehenen Aufrufe für die Tasksteuerung und Synchronisation bearbeitet werden kann. Darüberhinaus wurden jedoch Bausteine realisiert, die über die von PEARL zur Verfügung gestellten Sprachmittel (bzw. über die den Sprachmitteln unterlegte Semantik) hinausgehen. Dies wurde notwendig, da in PEARL für die Zuteilung von Betriebsmitteln nur eine über Prioritätsnummern gesteuerte Strategie vorgesehen ist.

Es wurde vorausgesetzt, daß sich das Betriebssystem zusammen mit allen Anwenderprogrammen und Daten resident im Zentralspeicher befindet. Ein Datenverkehr ist nur zwischen Anwendertasks und Standard- bzw. Prozeßgeräten vorgesehen, wobei Formatierung und Ähnliches auf Anwenderebene (z.B. in geeigneten Bibliotheksroutinen) vorgenommen werden muß. Periphere Speicher werden nicht berücksichtigt. Die Anzahl der möglichen Anwendertasks sowie von Synchronisationsvariablen (Semaphore, Bolts) ist theoretisch beliebig groß. Ein Großteil des notwendigen Speicherplatzes für Verwaltungsdaten - Taskkontrollblöcke, Schedules, etc. - muß im Anwenderprogramm zur Verfügung gestellt werden.

In der Folge sind jene PEARL-Anweisungen angeführt, die das Modellbetriebssystem verarbeiten kann. In eckige Klammern eingeschlossene Satzteile sind optional. Alternativen sind durch Senkrechtstriche getrennt und gemeinsam in geschweifte Klammern eingeschlossen. Kleingeschriebene Worte erklären sich bei Kenntnis der Sprache PEARL selbst. Soweit diesen Begriffen auf Sprachebene nicht Variablen oder Konstanten sondern Ausdrücke entsprechen, wird vorausgesetzt, daß diese vor Anstoß des Betriebssystemaufrufes bereits ausgewertet sind. Großgeschriebene Worte stellen Schlüsselwörter dar. Modifikationen gegenüber PEARL sind jeweils durch Bemerkungen erklärt [PDV77] .

#### Aktivierungsanweisung:

```
[WHEN interrupt] [AFTER duration] [ALL duration] ACTIVATE task  
[ (PRIORITY integer | EXECTIME duration) ];
```

Funktion: Die angegebene Task wird (falls sie sich im Zustand "ruhend" befindet) in den Zustand "bereit" versetzt (siehe Bild 4.1). Dabei kann ihr eine Prioritätszahl (PRIORITY...) bzw. eine maximale Antwortzeit (EXECTIME...) zugeteilt werden.

Unterbleibt diese Angabe, so wird der Task die niedrigste Dringlichkeitsstufe zugewiesen. Bei Angabe einer Einplanungsbedingung erfolgt der Zustandswechsel erst beim Eintreffen eines Interrupts (WHEN...), nach Verstreichen einer Zeitspanne (AFTER...), bzw. zyklisch (ALL...).

Bemerkungen: Die Schedule-Variante WHEN-AFTER... ist im Unterschied zu PEARL erlaubt. Statt einer Priorität kann auch eine maximale Antwortzeit vorgegeben werden - innerhalb eines Programmsystems kann jedoch nur jeweils eine der beiden Alternativen gewählt werden.

#### Unterbreuchungsanweisung:

{WHEN interrupt | [WHEN interrupt] AFTER duration} RESUME  
[ {PRIORITY integer | EXECTIME duration} ];

Funktion: Die aufrufende Task wird in den Zustand "unterbrochen" versetzt. Erst wenn die angegebene Einplanungsbedingung erfüllt ist, wird sie wieder in den Zustand "bereit" versetzt.

Bemerkungen: Es gelten die Bemerkungen zur Aktivierungsanweisung analog.

#### Beendigungsanweisung:

TERMINATE [task];

Funktion: Die angegebene (bzw. die aufrufende Task) wird in den Zustand "ruhend" versetzt.

Bemerkungen: Die Beendigung einer Task erfolgt u.U. nicht unmittelbar. Falls sie zum Zeitpunkt des Aufrufes noch Bolt-Variable blockiert, wird sie veranlaßt, diese vorher freizugeben.

#### Prioritätsänderung:

CONTINUE [task] {PRIORITY integer | EXECTIME duration};

Funktion: Die Priorität oder die Restzeit der angegebenen (bzw. der aufrufenden) Task wird verändert. Diese Änderung wird nur wirksam, wenn sich die betreffende Task nicht im

Zustand "ruhend" befindet.

Bemerkungen: Die Anweisung bewirkt keine Fortsetzung einer unterbrochenen Task.

#### Semaphor-Anweisungen:

REQUEST semaphor;

Funktion: Auf die angegebene Semaphor-Variable wird eine "P-Operation" angewandt.

[WHEN interrupt] [AFTER duration] [ALL duration] RELEASE semaphor;

Funktion: Auf die angegebene Semaphor-Variable wird eine "V-Operation" angewandt bzw. eingeplant.

Bemerkungen: Im Unterschied zu PEARL darf eine RELEASE-Anweisung eingeplant werden.

#### Bolt-Anweisungen:

RESERVE bolt;

Funktion: Die angegebene Bolt-Variable wird belegt.

FREE bolt;

Funktion: Die angegebene Bolt-Variable wird freigegeben.

Bemerkungen: Die Bolt-Anweisungen werden zur Realisierung von "kritischen Abschnitten" verwendet. Das Betriebssystem prüft daher auf die jeweils "symmetrische" Anwendung der beiden Aufrufe innerhalb einer Task.

#### Löschen von Einplanungen:

PREVENT { [task] | semaphor };

Funktion: Löschen der Einplanung einer Aktivierung der angegebenen (bzw. der aufrufenden) Task oder Löschen der Einplanung einer RELEASE-Operation auf die angegebene Semaphor-Variable.

Bemerkungen: Eine PREVENT-Anweisung bezüglich einer Semaphor-

variablen löscht eine etwaige RELEASE-Einplanung. Eine RESUME-Einplanung kann nicht explizit gelöscht werden.

#### Ein/Ausgabe-Anweisungen:

Der anlagenunabhängige Teil des Betriebssystems versteht nur den Auftrag, daß ab einer bestimmten Speicheradresse Daten zu übertragen sind. Er organisiert die Gerätewarteschlangen und veranlaßt die Fortsetzung der aufrufenden Task nach der Fertigmeldung des Gerätes. Für jedes Gerät ist eine anlagenabhängige Befehlsfolge zur Verfügung zu stellen, die den eigentliche Anstoß der Operation bewirkt. Sämtliche Modifikationen der zu transferierenden Daten (Formatierung, Kalibrierung, etc.) sind innerhalb der Anwendertask vorzunehmen.

Bild 4.1 zeigt die möglichen Taskzustände sowie die zulässigen Zustandsübergänge. Die angegebenen Zahlen haben die folgenden Bedeutungen:

- 1 Aktivierung (ACTIVATE)
- 2 Beendigung (TERMINATE)
- 3 Prozessorzuteilung durch Prozessorverwaltung
- 4 Unterbrechung (RESUME oder E/A-Anstoß)
- 5 Fortsetzung durch RESUME-Einplanungsbedingung oder Geräte-Fertigmeldung
- 6 Blockierung durch Bolt oder Semaphore (REQUEST, RESERVE)
- 7 Freigabe (RELEASE, FREE)
- 8 Verdrängung (durch höherpriorie Task)

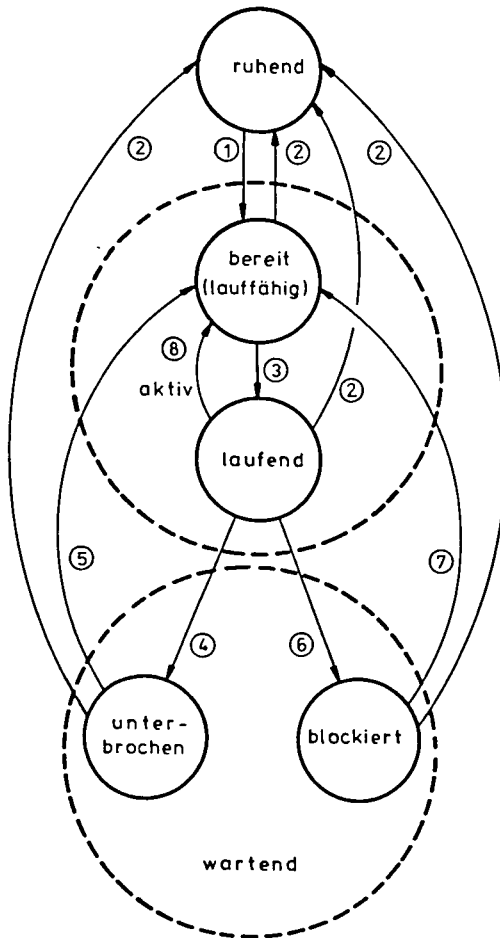


Bild 4.1: Taskzustandsdiagramm

#### 4.1.2. Implementationstechnik

Für die Programmierung des Modellbetriebssystems wurde eine eigens dafür entwickelte Programmiersprache verwendet, die etwas über Assemblerniveau liegt [ROE77]. Die Fähigkeiten der zugrunde gelegten virtuellen Maschine wurden von modernen Einprozessoranlagen abgeleitet (PDP-11, SIEMENS-330, etc.), sodaß eine effektive Übersetzung in den Zielcode dieser Maschinen möglich ist. Diese Methode wurde ausgewählt, weil an der verwendeten Anlage (SIEMENS-306) keine geeignete anlagenunabhängige Programmiersprache zur Verfügung stand. Andere Maschinen kamen aber unter anderem wegen des Fehlens eines für die Programmierung der Anwenderprogramme notwendigen PEARL-Übersetzers nicht in Frage. Trotzdem erlaubte das Verfahren, etwa 80-90% des Betriebssystems anlagenunabhängig zu formulieren. Bei der verwendeten Zielmaschine SIEMENS-306 (auf die die Implementationssprache nicht optimal zugeschnitten ist), belegt der anlagenunabhängige Teil (ohne Listen) etwa 2000 und der anlagenabhängige Teil etwa 300 Worte. Letzterer besteht hauptsächlich aus geräteabhängigen E/A-Routinen sowie aus Befehlsfolgen, die die bei der Unterbrechungs- bzw. Aufrufbearbeitung anfallenden Parameter in einer Form aufbereiten, daß die im anlagenunabhängigen Teil spezifizierten Schnittstellen befriedigt werden. Ein primitiver Lader sowie ein relativ komfortables Testpaket für den anlagenunabhängigen Teil wurden zusätzlich in Assembler programmiert.

Für die Übersetzung der Implementationssprache in SIEMENS-300-Assembler ("PROSA-300") wurde der Makro-Prozessor STAGE-2 [WAI70] verwendet. Etwa 90 Makros waren erforderlich, wobei ein Teil des Aufwandes darauf zurückzuführen ist, daß das Format der Sprache möglichst dokumentationsfreundlich (d.h. frei) gewählt wurde.

#### 4.2. Grundsätzlicher Aufbau

Das Modellbetriebssystem kann grob in die folgenden 6 Verwaltungsbausteine gegliedert werden:

- . Auftragsübernahme (Schnittstelle zu Anwendertasks)
- . Taskverwaltung (Abwicklung von "Tasking"- und Synchronisationsaufrufen)
- . Ein/Ausgabeverwaltung (Verwaltung von Standard- und Prozessperipherie)
- . Einplanungsverwaltung (Verwaltung von Einplanungsbedingungen - "Schedules")
- . Unterbrechungsverwaltung (Bearbeitung von Zeittakt, Alarmsignal, Fertigmeldungen der Geräte)
- . Prozessorverwaltung (Zuteilung des Prozessors und "Start" der Tasks)

Die Zusammenarbeit der Verwaltungsbausteine ist in Bild 4.2 dargestellt.

Anwendertasks übergeben alle Betriebssystemaufrufe an die AUFTRAGSÜBERNAHME (1). Diese stößt auf Grund der Aufrufparameter einen Modul der TASKVERWALTUNG (2) oder der EIN/AUSGABEVERWALTUNG (3) an. Anschließend wird die PROZESSORVERWALTUNG (4) beauftragt, die Kontrolle über den Prozessor wieder an eine Anwendertask zu übergeben (5). Die TASKVERWALTUNG bewirkt entweder unmittelbar die vorgesehenen Zustandsänderungen, oder veranlaßt die Einplanung derselben über einen Aufruf der EINPLANUNGSVERWALTUNG (6). Die EIN/AUSGABEVERWALTUNG veranlaßt eine Zurückstellung der aufrufenden Task und die Einleitung des Transfers über das ausgewählte Gerät, bzw. puffert den entsprechenden Auftrag, falls dieses bereits belegt ist. Die UNTERBRECHUNGSVERWALTUNG stellt beim Eintreffen eines Alarmsignals bzw. Zeittaktes fest, ob für das betreffende Ereignis Einplanungen vorliegen und ruft in diesem Falle die EINPLANUNGSVERWALTUNG auf (7). Diese meldet zurück, ob die Bedingungen zur Ausführung einer (und welcher) Zustandsänderung bereits vollständig erfüllt sind, worauf der entsprechende Auftrag der TASKVERWALTUNG übergeben werden kann (8). Nach Bearbeitung aller möglichen Einplanungen wird abschließend die PROZESSORVERWALTUNG angestoßen (9). Die UNTERBRECHUNGSVERWALTUNG erhält außerdem Fertigmeldungen von Geräten, die an die EIN/AUSGABEVERWALTUNG

weitergereicht werden (10) . Die EIN/AUSGABEVERWALTUNG führt Transfer-Abschlußroutinen durch, wobei unter anderem geprüft wird, ob weitere Aufträge für das entsprechende Gerät vorliegen. In diesem Fall erfolgt ein erneuter Transfer-Anstoß. Abschließend wird die PROZESSORVERWALTUNG mit der Fortsetzung einer Anwendertask beauftragt (11) .

Eine detaillierte Beschreibung der Verwaltungsbausteine sowie der notwendigen Datenstrukturen findet man im Anhang II.

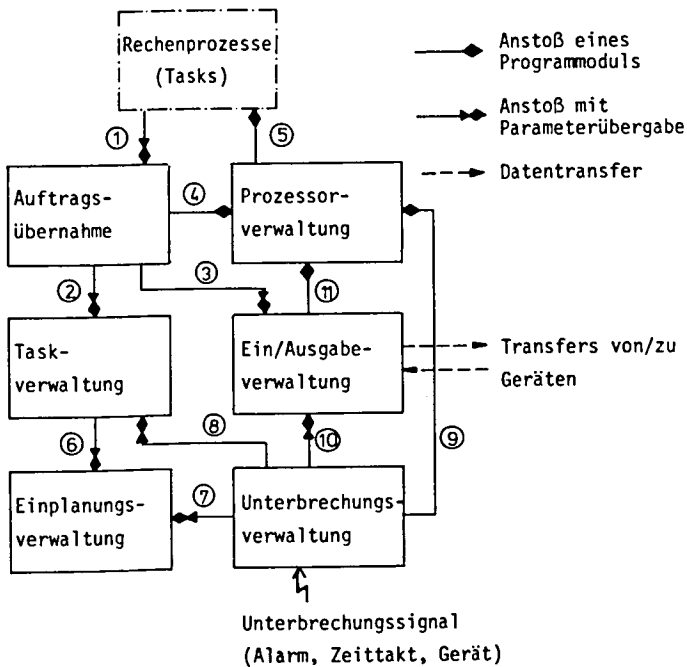


Bild 4.2: Schema der Zusammenarbeit der Verwaltungsbausteine



#### 4.3. Realisierung der Taskverwaltungsstrategien

Bei der Implementation des Modellbetriebssystems wurde darauf geachtet, daß mit möglichst geringem Umstellungsaufwand von einer Strategie zu einer anderen übergegangen werden konnte. Dies hatte natürlich eine Erhöhung des Laufzeitaufwandes für jene Strategien zur Folge, die für sich allein mit extrem einfachen Techniken abgewickelt werden könnten ("FIFO", "FP"). Geht man jedoch davon aus, daß der Zeitaufwand für Verwaltungsaufgaben im Betriebssystem trotzdem noch klein gegenüber den Laufzeiten von Anwendertasks ist, so fällt dieser Einfluß für vergleichende Untersuchungen kaum ins Gewicht.

Grundsätzlich werden bei Konkurrenzsituationen um Betriebsmittel (Prozessor, Geräte, Synchronisationsvariable) Warteschlangen verwendet (siehe Kap.3.2.1., Bild 3.8). Die Elemente dieser Warteschlangen - die Taskverwaltungsblöcke - sind ausschließlich nach Prioritätszahlen geordnet.

##### 4.3.1. Strategie "First-in-first-out" ("FIFO")

Da beim Einketten in eine Warteschlange immer vor den einzuketten- den Taskverwaltungsblock alle jene zu liegen kommen, die höhere oder gleiche Priorität besitzen, entsteht bei Verwendung gleicher Prioritäten für alle Tasks automatisch die Strategie "FIFO".

Ein Hinzufügen eines neuen Elementes erfolgt ja dann immer an das Ende, ein Wegnehmen hingegen vom Anfang der Warteschlange.

Da der Suchalgorithmus für das Einketten immer beim Warteschlangenkopf beginnt, entsteht ein im Grunde genommen überflüssiger Laufzeitaufwand, weil das Warteschlangende auch bekannt ist. Dies wurde jedoch wegen der angestrebten Einheitlichkeit der Implementationstechnik in Kauf genommen.

Die Forderung "gleiche Priorität für alle Tasks" wird am einfachsten dadurch befriedigt, daß überhaupt keine Prioritäten angegeben werden. Das Betriebssystem setzt dann ersatzweise immer die gleiche (niedrigste) Priorität ein.

#### 4.3.2. Feste Prioritäten ("FP")

Da das vorliegende Konzept die volldynamische Vergabe von Prioritäten ermöglicht, ist die Nachbildung der Strategie "FP" trivial. Es muß nur im Anwenderprogramm darauf geachtet werden, daß alle Startaufrufe, die sich auf dieselbe Task beziehen, die gleichen Prioritätsangaben enthalten.

#### 4.3.3. Strategie nach der kürzesten Restzeit ("KR")

Die Strategie "KR" wurde dadurch realisiert, daß eine vorgegebene maximale Antwortzeit durch Addition der aktuellen Zeit in eine Prioritätszahl umgerechnet wird. Diese Summe stellt ja die Zeitschranke dar, die selbst keine Funktion der Zeit - also eine Konstante ist.

Das einzige Problem besteht darin, daß die Darstellung der aktuellen Zeit im Rechner in gewissen Abständen immer wieder mit einem Anfangswert beginnen muß. Dies hat zur Folge, daß der beim Einketten in Warteschlangen (und auch an anderen Stellen) notwendige Prioritätsvergleich die Tatsache berücksichtigen muß, daß nicht immer größere Zahlenwerte kleinere Prioritäten repräsentieren. Diese Schwierigkeit wurde mit der bereits in Kap.3.2.4. beschriebenen Vorgangsweise behoben.

Beim Übergang auf andere Strategien, die nicht restzeitorientiert sind, braucht nur die Hochzählung der erwähnten Zeitzelle zu unterbleiben. Der Vergleichsmechanismus für Prioritätszahlen kann unverändert bleiben.

#### 4.3.4. Zusatzbedingung "Restzeitreduktion"

Die Zusatzbedingung "Restzeitreduktion" erfordert keine zusätzlichen Maßnahmen im Betriebssystem. Die Reduktion und Wiederanhebung der Restzeit (bzw. der Zeitschranke) erfolgt in den Anwendertasks unter Zuhilfenahme der für Prioritätsänderungen vorgesehenen CONTINUE-Anweisung.

#### 4.3.5. "Kritische Prioritäten"

Die "kritischen Prioritäten" stellen ein Konzept dar, das zusätzlich zu den erwähnten anderen Strategien (außer "FIFO") angewandt werden kann. Die zugrundeliegenden Gedanken wurden bereits in Kap. 3.1.8. beschrieben und sollen hier kurz wiederholt werden (ein kritischer Abschnitt wird im Modellbetriebssystem durch die Belegung bzw. Freigabe einer "Boltvariablen" realisiert):

Eine Task, die sich in einem kritischen Abschnitt befindet, blockiert dadurch unter Umständen andere "wichtigere" Tasks (Tasks mit höherer Priorität bzw. kürzerer Restzeit).

Es liegt nahe, eine derartige "Blockierungsursache" mit der gleichen Priorität zu beseitigen, die der wichtigsten, durch sie blockierten Task eigen ist. Dazu ist es erforderlich, im Taskverwaltungsblock neben der "eigentlichen" Priorität (Originalpriorität) eine "kritische" Priorität (aktuelle Priorität) zu führen, die innerhalb eines kritischen Abschnittes unter den genannten Umständen höher sein kann als die Originalpriorität.

Bei einer Verschachtelung von kritischen Abschnitten ergibt sich die Notwendigkeit einer Prioritätsüberprüfung über mehrere Ebenen hinweg. Die dafür notwendigen Algorithmen bedienen sich eines sogenannten "Boltbelegungsgraphen", der den Belegungszustand aller Boltvariablen beschreibt. Dieser Boltbelegungsgraph ist im Modellbetriebssystem durch die Prozessorwarteschlange, die Schlangen der auf den Zugriff auf eine Boltvariable wartenden Tasks (Bolt-Warteschlange) sowie durch spezielle Ketten, die alle durch jeweils eine Task belegten Bolt-Variablen verketteten, repräsentiert.

Im Bild 4.3 ist ein "Boltbelegungsgraph" dargestellt. Er zeigt eine zur Laufzeit eines Programmsystems sich einstellende mögliche Verknüpfung zwischen Taskverwaltungsblöcken und Bolt-Verwaltungsblöcken. Die waagrechten Pfeile verbinden die um den Prozessor konkurrierenden Tasks, bzw. die Tasks, die auf die Freigabe einer Boltvariablen warten. Sie repräsentieren die nach Priorität geordnete Prozessorwarteschlange bzw. die Bolt-Warteschlangen. Die schräg nach unten führenden Pfeile verbinden die von einer Task belegten Boltvariablen, wobei die Belegungsreihenfolge entgegengesetzt der Pfeilrichtung zu denken ist ("Task-Bolt-Kette").

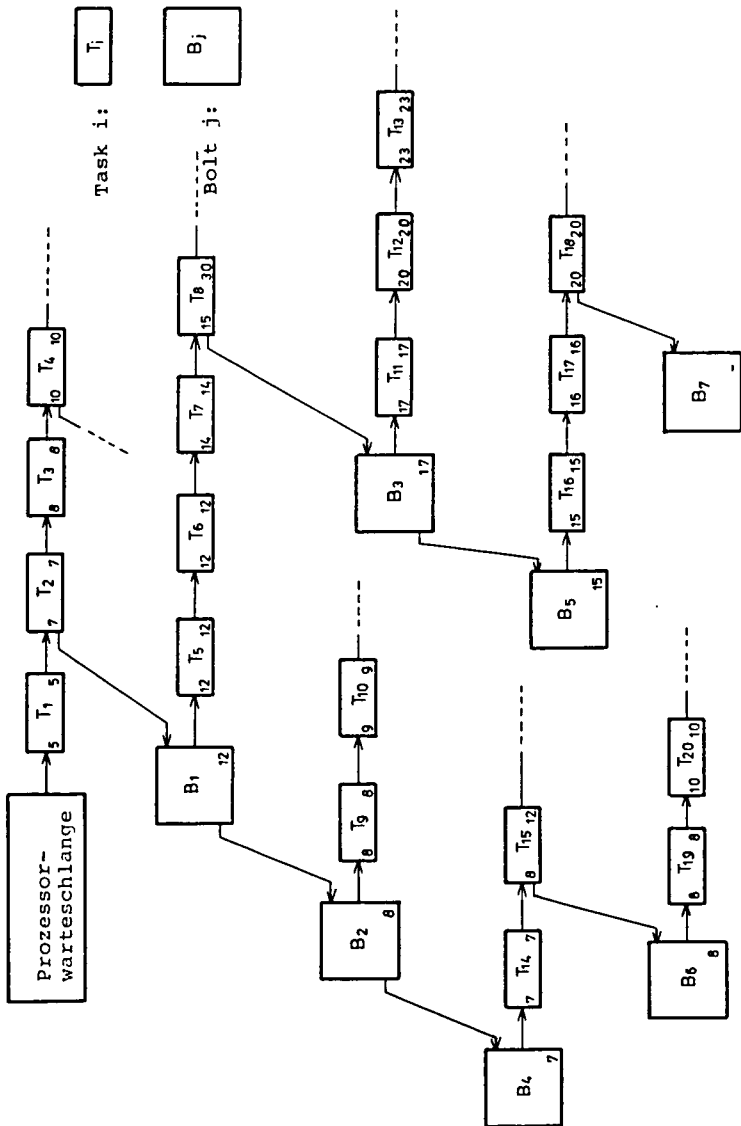


Bild 4.3: Bolt-Belegungsgraph

Eine Task gibt also immer zuerst die mit ihr unmittelbar verbundene Boltvariable frei.

Ein derartiges (schleifenfreies) Belegungsschema stellt immer einen verklemmungsfreien Zustand dar. Eine Verklemmung würde z.B. dann entstehen, wenn die Task  $T_2$  die Boltvariable  $B_3$  reservieren wollte: Da bereits ein Weg von  $T_2$  nach  $B_3$  existiert, wäre die Darstellung in Baumform dann nicht mehr möglich (vgl. [KRA75], Seite 86).

Damit ist der Weg zu einem Algorithmus angedeutet, der zur Laufzeit eine Systemverklemmung erkennt und ein Signal an die Task erzeugen kann, die die Verklemmung verursacht.

Es ist klar, daß eine Task für alle im Graphen "weiter unten" angeordneten Tasks ein Laufhindernis darstellt: Sie muß erst mindestens eine Boltvariable wieder freigeben, ehe die Chance besteht, daß eine der "behinderten" Tasks fortgesetzt werden kann. Hat nun eine "weiter unten" befindliche Task eine hohe Priorität, so ist es sinnvoll zu fordern, daß die Priorität aller "über" ihr angeordneten Tasks auf das gleiche Niveau angehoben wird. Dies bedeutet: Ein Laufhindernis für eine Task wird mit mindestens der gleichen Priorität beseitigt, mit der die Task selbst ausgeführt werden soll.

Jeder Task sind demnach 2 Prioritätszahlen zugeordnet: Eine Originalprioritätszahl  $\overline{p}_0$  und eine "kritische" (aktuelle) Prioritätszahl  $\overline{p}_k$ , wobei gilt  $\overline{p}_k \leq \overline{p}_0$  (kleinere Zahlen stellen höhere Prioritäten dar).

Im Bild 4.3 stellen die Zahlen links unten (in den Taskverwaltungsblöcken) die kritischen, und rechts unten die Originalprioritätszahlen dar. Die Zahlen in den Boltverwaltungsblöcken stellen "Boltprioritätszahlen" dar. Sie sind identisch mit den kritischen Prioritätszahlen der ersten Tasks in den zugehörigen Boltwarteschlangen. Im konkreten Beispiel haben die Tasks  $T_8$  und  $T_{15}$  eine angehobene kritische Priorität, da sie höherpriorie Tasks ( $T_{16}$  und  $T_{19}$ ) behindern. Der Algorithmus des Anhebens der Prioritäten sei durch die Task  $T_1$  demonstriert, deren Prioritätszahl 5 ist und die die Boltvariable  $B_6$  anfordere:

- .  $T_1$  wird (nach Priorität) vor  $T_{19}$  eingekettet
- . Die Boltpriorität von  $B_6$  ändert sich zu 5

- . Die (kritische) Prioritätszahl der darüberliegenden Task  $T_{15}$  ergibt sich zu 5
- .  $T_{15}$  tauscht mit  $T_{14}$  den Platz
- . Die Prioritätszahl von  $B_4$  ergibt sich zu 5
- . Die Prioritätszahl von  $T_2$  ergibt sich zu 5

Damit entsteht der in Bild 4.4 dargestellte Boltbelegungsgraph.

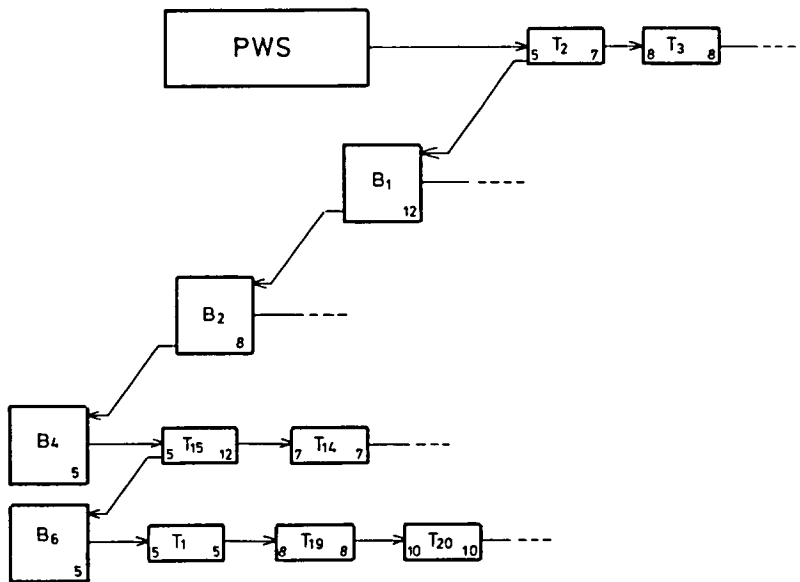


Bild 4.4: Boltbelegungsgraph aus Bild 4.3 nach Anforderung von  $B_6$  durch die Task  $T_1$

Prioritätsanhebungen müssen natürlich im Zuge der entsprechenden Freigabeoperationen wieder rückgängig gemacht werden. Da eine Freigabe nur durch eine laufende Task erfolgen kann, sei der Vorgang am Beispiel der Task  $T_2$  demonstriert:  $T_2$  gibt zuerst  $B_1$  und  $B_2$  frei. Dabei erfolgt keine Zurückstufung der Priorität, da der Teilbaum ab  $B_4$  mit der Prioritätszahl 5 sich noch "unterhalb" von  $T_2$  befindet.

Diesen Zustand zeigt Bild 4.5. Erst wenn  $B_4$  freigegeben wird, kann die Originalpriorität von  $T_2$  wieder eingesetzt werden, da jetzt  $T_1$  nicht mehr durch  $T_2$  behindert ist (siehe Bild 4.6).

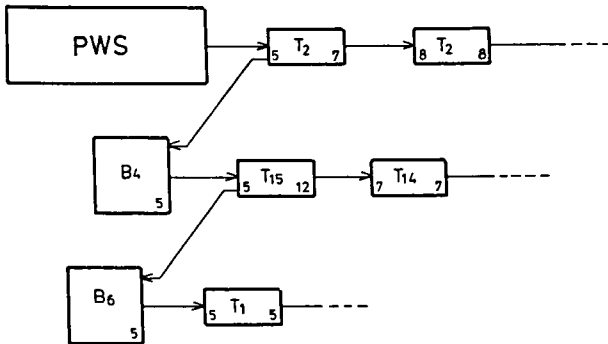


Bild 4.5: Boltbelegungsgraph aus Bild 4.4 nach Freigabe von  $B_1$  und  $B_2$  durch  $T_2$

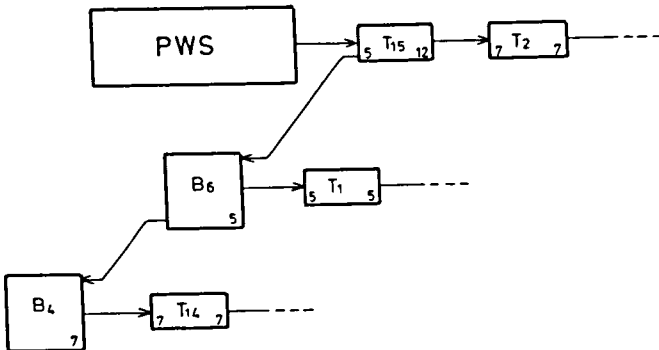


Bild 4.6: Boltbelegungsgraph aus Bild 4.5 nach Freigabe von  $B_4$  durch  $T_2$

Zusammenfassend kann gesagt werden: Bei allen Operationen, die eine Änderung der Belegungssituation zur Folge haben, muß dafür Sorge getragen werden, daß die kritische (aktuelle) Prioritätszahl einer Task  $T_i$  sich ergibt zu

$$\overline{p_{k_i}} = \min \left\{ \overline{p_{o_i}}, \min \left\{ \overline{p_{k_j}} \mid \text{für alle } T_j \text{ "unterhalb" } T_i \right\} \right\}$$

Die dafür implementierten Algorithmen werden hier nicht im Detail beschrieben.



## 5. PRAKTIISCHE ERPROBUNG VON TASKVERWALTUNGSSTRATEGIEN

Um die in den vorangegangenen Kapiteln dargelegten theoretischen Erkenntnisse an praktischen Beispielen überprüfen zu können, wurde ein Weg gesucht, die im Modellbetriebssystem implementierten Strategien ohne softwaremäßige Prozeßsimulation - also unter möglichst "echten" Bedingungen zu testen. Ein geeigneter Modellprozeß hätte vor allem der Forderung nach Modifizierbarkeit der zeitlichen Anforderungen genügen müssen, um einen Vergleich der Strategien unter verschiedenen Belastungen zu ermöglichen. Da der Aufwand für den Entwurf und den Aufbau eines solchen Modellprozesses den Umfang dieser Arbeit gesprengt hätte, mußte darauf verzichtet werden. Ein für das angestrebte Ziel nahezu idealer Ersatz stand jedoch bereits in der Form des am Institut für Regelungstechnik und Prozeßautomatisierung der Universität Stuttgart entwickelten "Prozeßrechnertestgerätes" zur Verfügung. Mit Hilfe dieses Gerätes konnte die Kommunikation zwischen Programmsystem und technischem Prozeß hardwaremäßig simuliert werden, wobei zusätzlich eine Überwachung der zeitlichen Vorgänge im Rechner möglich war.

Zum Verständnis der weiteren Ausführungen wird im folgenden Abschnitt die Funktionsweise des Prozeßrechnertestgerätes umrissen.

### 5.1. Funktionsweise des Prozeßrechnertestgerätes [BAE76,KUE76,LAU77]

Das Prozeßrechnertestgerät wurde zum Zwecke der Bestimmung der Leistungsfähigkeit von Prozeßrechnern entworfen. Durch einen Mikroprozessor INTEL 8080 werden auf bis zu 4 Kanälen Interrupts erzeugt, die in dem zu testenden Prozeßrechner den Start von entsprechenden Reaktionsprogrammen bewirken (können). Gleichzeitig mit der Erzeugung eines Interrupts wird ein zugehöriger "Reaktionszeitzähler" gestartet, d.h. mit einem Takt wählbarer Frequenz beaufschlagt. Es wird solange gezählt, bis vom Prozeßrechner eine Digitalausgabe erfolgt, die den "Reaktionszeitzähler" stoppt und gleichzeitig einen "Ausführungszeitzähler" startet, der schließlich mit einer zweiten Digitalausgabe ebenfalls wieder gestoppt werden muß. Damit ist z.B. die Zeit bis zur Erkennung eines Interrupts und bis zur vollständigen Abarbeitung eines zugehörigen Unterbrechungsantwortprogrammes bestimmbar. In Anlehnung an DIN 66216 (siehe auch

Bild 2.1) sind in Bild 5.1 Zeitintervalle dargestellt, wie sie bei der Anwendung des Prozeßrechnertestgerätes in diesem Sinne auftreten können, wobei angenommen ist, daß zum Zeitpunkt des Auftretens des Interrupts ein Programm tätig ist, das eine geringere Priorität als das Unterbrechungsantwortprogramm hat, und daß die Digitalausgaben zum Stoppen der Zähler im Unterbrechungsantwortprogramm als erste bzw. letzte Anweisung erfolgen.

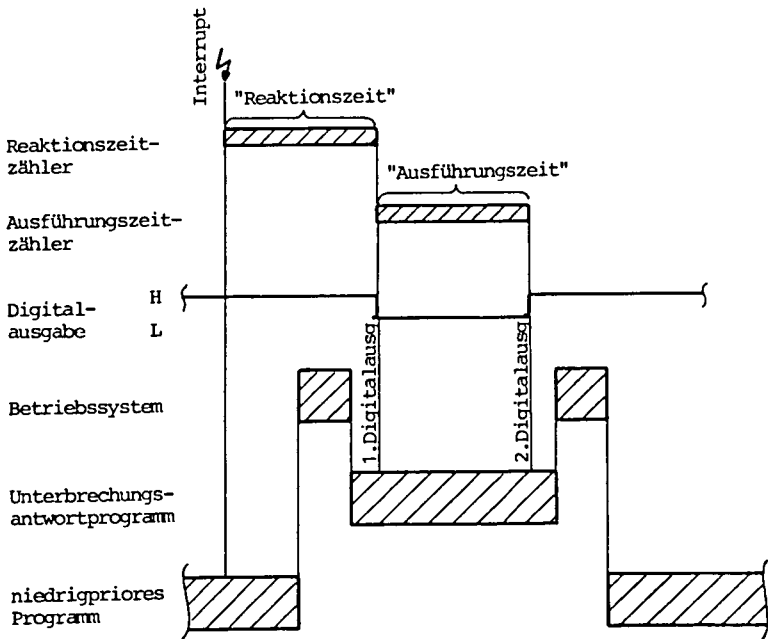


Bild 5.1: Schema der charakteristischen Zeitintervalle beim Einsatz des Prozeßrechnertestgerätes

Um zu vermeiden, daß bei kurz hintereinander auftretenden Interrupts ein Meßwert verloren geht, ist für jeden Kanal das Paar "Reaktionszeit-zähler" - "Ausführungszeit-zähler" doppelt vorhanden, sodaß eine Umschaltung möglich ist. Sind bei der Erzeugung eines Interrupts jedoch bereits beide Zählerpaare belegt, so wird der

Prozeßrechner als überlastet betrachtet und es werden keine weiteren Meßwerte für diesen Interrupt aufgenommen.

Die Digitalausgaben vom Prozeßrechner veranlassen neben dem Stoppen der Zähler ein Auslesen von "Reaktionszeit" bzw. "Ausführungszeit" in den Datenspeicher des Mikroprozessors durch ein schnelles Steuerwerk, wobei jedem Interrupt ein zusammenhängender Speicherbereich von 512 Bytes zugeordnet ist (für je 2 mal 128 Daten a 16 bit). Es können demnach auf jedem Kanal bis zu 128 Interrupts erzeugt und die entsprechenden Zeitmessungen vorgenommen werden.

Die Festlegung des Interruptschemas und eine anschließende Auswertung der Meßwerte erfolgt durch Programme, die im PROM des Mikroprozessors liegen, und die per Dialog über Teletype bedient werden können. Zum Zeitpunkt der Verwendung des Prozeßrechnertestgerätes für diese Arbeit standen ein Programm zur Vorwahl eines zyklischen Interruptschemas sowie ein Programm zur Vorwahl einer quasi-stochastischen Interruptfolge zur Verfügung. Die Auswertung erfolgte in beiden Fällen auf gleiche Weise, nämlich durch Ausgabe von Minimalwert, Mittelwert und Maximalwert der gemessenen Zeiten, sowie durch wahlweise Protokollierung aller Meßwerte in geordneter bzw. ungeordneter Form. Eine ausführliche Beschreibung ist der einschlägigen Literatur zu entnehmen [BAE76,KUE76,LAU77].

## 5.2. Vorgangsweise

Als Unterbrechungsantwortprogramme wurden Tasks im Sinne von PEARL vorgesehen, die durch "WHEN-Schedules" mit den betreffenden Interrupts verknüpft wurden. Als Bewertungskriterium für die Leistungsfähigkeit der Taskverwaltungsstrategien bot sich ein Vergleich der erzielten Antwortzeiten an, was darauf hinauslief, daß vor allem die Summe von Reaktionszeit und Ausführungszeit von Interesse war. Es wurden daher bei fast allen Versuchen die zum Stoppen von Reaktionszeitzähler und Ausführungszeitzählernotwendigen Digitalausgaben unmittelbar hintereinander vor dem Ende der Antworttasks vorgesehen. Damit wurde statt der Reaktionszeit direkt die Antwortzeit gemessen und die gemessene "Ausführungszeit" war ohne Bedeutung. Bild 5.2 zeigt schematisch die Zeitintervalle einer möglichen Konstellation. Es ist hier angenommen, daß zum Zeitpunkt des Auftretens eines Interrupts eine Task läuft, die höhere Priorität hat als die

betreffende Antworttask.

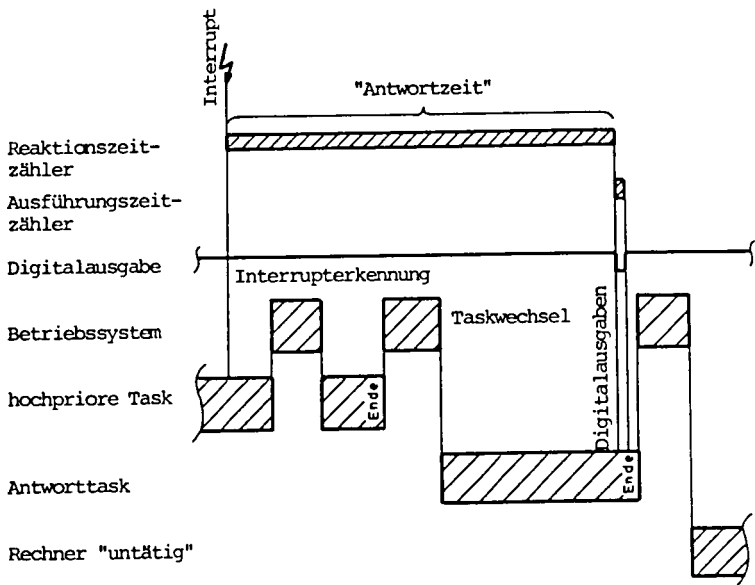


Bild 5.2: Schema der charakteristischen Zeitintervalle bei der Erprobung der Taskverwaltungsstrategien mit dem Prozeßrechner-testgerät

Es wurde bei allen Versuchen die volle Zahl von 4 Interrupts ausgenutzt, d.h. es wurden jeweils 4 Antworttasks vorgesehen. Eine weitere Task ermöglichte im Dialog die Modifikation der Aktivitäten der Antworttasks, die Auswahl der verwendeten Taskverwaltungsstrategie, sowie die Vorgabe von Zeitschranken bzw. Prioritäten.

Die möglichen Ablaufalternativen der Antworttasks sind aus dem Struktogramm in Bild 5.3 ersichtlich. Dazu waren die folgenden Parameter als bedienbare Größen vorgesehen:

- . Zeitdauer der 1.Prozessorbelegung ( $t_{1i}$  msec)
- . Zeitdauer der 2.Prozessorbelegung ( $t_{2i}$  msec)

- . Wartezeit ( $w_i$  msec)
- . Analogeingabe ja/nein (boole'sche Variable  $a_i$ )
- . Kritischer Abschnitt ja/nein (boole'sche Variable  $k_i$ )

Die Auswahl der aktuellen Taskverwaltungsstrategie wurde wie folgt vorgenommen:

- . Die Entscheidung zwischen prioritätsgesteuerter bzw. restzeitgesteuerter Strategie durch Umschalten der dafür vorgesehenen Weiche im Modellbetriebssystem
- . Die Zusatzbedingung "Restzeitreduktion" durch Vorgabe einer um die Wartezeit  $w_i$  bzw. die Dauer der Analogeingabe reduzierten Zeitschranke bei der Task-einplanung, sowie nach erfolgter Wartezeit bzw. Analogeingabe durch eine Wiederanhebung der verbleibenden Restzeit um den gleichen Betrag
- . Die Zusatzbedingung "Kritische Priorität" durch Verwendung einer "BOLT-Variablen" zur Synchronisation (Im Normalfall erfolgte die Synchronisation über eine "SEMAPHOR"-Variable, für die im Betriebssystem der Mechanismus zur Realisierung der kritischen Prioritäten nicht implementiert ist)

Diese Maßnahmen, sowie die Vorgabe von Prioritäten bzw. Zeitschranken wurden über dynamische Modifikationen von Daten bzw. Code im Modellbetriebssystem oder in den Antworttasks vollzogen und konnten demnach nicht in PEARL formuliert werden, sondern mußten durch Assemblereinschübe bewerkstelligt werden.

Der Zeittakt des Rechners betrug 10 Millisekunden, sodaß für die in die Taskverwaltung eingehenden Zeitangaben nur ein Vielfaches dieses Betrages angegeben werden konnte. Dies betraf insbesondere die vorgegebenen Zeitschranken, sowie die zur Restzeitreduktion herangezogenen Wartezeiten bzw. die Analogeingabezeit. Letztere betrug ca. 27 Millisekunden (20 Millisekunden Integrationszeit plus Schaltzeiten) und wurde deshalb auf 30 Millisekunden aufgerundet.

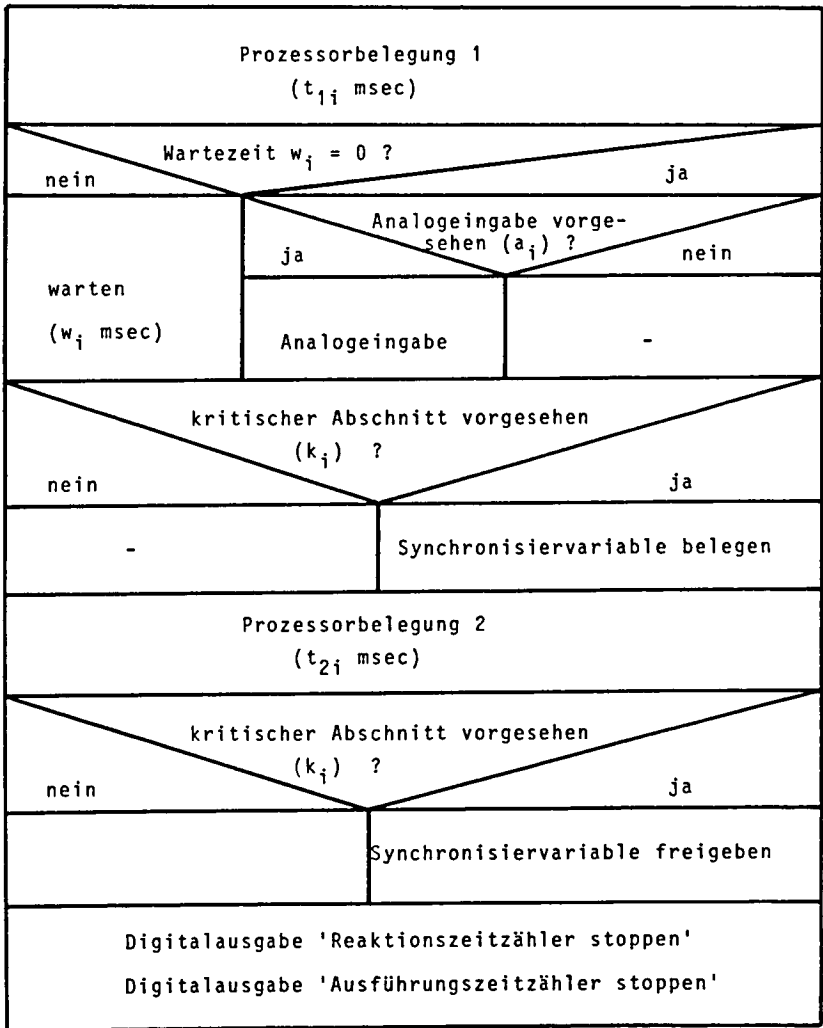


Bild 5.3: Schematischer Ablauf der Antworttask  $T_i$

Der Leistungsvergleich der Taskverwaltungsstrategien wurde in mehreren Versuchsreihen wie folgt vorgenommen:

- . Festlegung der Antworttaskabläufe
- . Festlegung eines Interruptschemas, bei dem alle zu vergleichenden Strategien zu einer Einhaltung der Zeitbedingungen führten, d.h. bei dem in keinem Fall eine Rechnerüberlastung auftrat
- . Allmähliche Verschärfung der Zeitbedingungen durch Verkürzung der Interruptabstände, sodaß nacheinander alle Strategien zu einer Rechnerüberlastung führten

Eine Überlastung des Rechners konnte bei einem zyklischen Interruptschema einfach erkannt werden: Da im Modellbetriebssystem keine Pufferung von Aktivierungen vorgesehen ist, ließ das Auftreten eines Meßwertes, der größer als der zugehörige Interruptzyklus war, darauf schließen, daß ein Interrupt verloren gegangen war. Damit waren aber auch alle folgenden Meßwerte (für den betreffenden Kanal) größer als der Zyklus bzw. waren gleich null, wenn noch ein zweiter Interrupt in der Folge nicht erkannt wurde. Bei stochastischen Interruptfolgen zeigte sich der Grad der Überlastung des Rechners darin, um wieviel ein Meßwertmaximum das zugehörige Minimum der Interruptabstände überschritt. Eine "echte" Überlastung - d.h. die Nichterkennung von Interrupts - war auch hier wieder durch einen größeren Sprung in der Meßwertreihe erkennbar.

Bei der Festlegung der Prioritäten bzw. Zeitschranken für die Beeinflussung der Taskverwaltungsstrategien wurde wie folgt vorgegangen: Bei Verwendung eines zyklischen Interruptschemas wurden die Prioritätszahlen proportional zu den Zykluszeiten gewählt (dies entsprach der Strategie "IFP" aus [SER72]), bzw. die Zeitschranken gleich den Zykluszeiten gesetzt. Bei stochastischen Interruptfolgen traten die minimalen Interruptabstände an die Stelle der Zyklen. Diese Vorgangsweise schien insofern sinnvoll zu sein, als der minimale Interruptabstand den "worst case" darstellte, und damit die potentiell größte "Wichtigkeit" der betreffenden Task widerspiegelte.

Ausschluß erfolgte. In diesem Fall wurde wieder eine Verbesserung der Verhältnisse durch "Restzeitreduktion" festgestellt.

Schließlich wurde in der 8. Versuchsreihe der bereits erwähnte Fall herbeigeführt, bei dem die Strategie "FP" gegenüber "KR" Vorteile bringt. Für diesen Zweck reichten 2 Antworttasks aus. Die eine Task belegte zuerst den Prozessor und anschließend die Analogeingabe, wogegen die Reihenfolge bei der anderen Task umgekehrt war. Die Interruptabstände waren zyklisch und die Task, der der größere Interruptabstand zugeordnet war, erhielt die höhere Priorität. Es muß hinzugefügt werden, daß bei einer Vorgabe von vertauschten und dadurch nicht zutreffenden Zeitschranken (entsprechend den nicht-"intelligenten" Prioritäten bei der Strategie "FP") auch die restzeitgesteuerten Strategien zu einer Einhaltung der Zeitbedingungen führten. Dies ist ein Hinweis darauf, daß es sinnvoll sein könnte, Restzeitreduktionen nur bei einem Teil der in Frage kommenden Tasks vorzunehmen, und zwar vorzugsweise bei solchen, deren Prozessorbelegungsphase vor der zur Reduktion herangezogenen Unterbrechung vergleichsweise kurz ist. Dies wäre im vorliegenden Fall die Task 2 gewesen, da diese vor der Analogeingabe nur kurzzeitig den Prozessor belegte.

Eine allgemein gültige Regel läßt sich jedoch kaum aufstellen. Es dürfte jedoch zweckmäßig sein, die Möglichkeit der Restzeitreduktion bei Verwendung der Strategie "KR" auf Sprachebene zur Verfügung zu stellen, um dem Anwender in kritischen Fällen noch eine Beeinflussungsmöglichkeit zu geben.

### 5.3.2. Erste Versuchsreihe

Zur Erläuterung der grundsätzlichen Vorgangsweise werden hier die einzelnen Versuchsschritte für die Versuchsreihe 1 im Detail beschrieben.

Mit dieser Versuchsreihe sollte die Überlegenheit der Strategie "KR" über die Strategie "FP", sowie die mögliche weitere Verbesserung des Zeitverhaltens durch "Restzeitreduktion" bei Tasks, die nicht ausschließlich den Prozessor belegen, demonstriert werden. Dazu wurden die Prozessorbelegungen zweier Tasks durch Wartezeiten von jeweils 40 Millisekunden unterbrochen. Die zeitlichen Abläufe aller 4 Tasks sind in Bild 5.4 dargestellt.



Task	1.Prozessorbelegung	Wartezeit	2.Prozessorbelegung (kein krit.Abschnitt)
1	3 msec	40 msec	2 msec
2	4 msec	40 msec	3 msec
3	15 msec	-	-
4	8 msec	-	-

Bild 5.4: Ablaufschema der Antworttasks bei der 1.Versuchsreihe

Es wurde ein zyklisches Interruptschema gewählt und mit den folgenden Zeiten begonnen:

Zyklus 1: 150 msec, Zyklus 2: 140 msec, Zyklus 3: 55 msec,  
Zyklus 4: 59 msec

Die Digitalausgaben wurden hier noch jeweils am Beginn und am Ende der Antworttasks vorgenommen, sodaß sowohl Reaktionszeit als auch Ausführungszeit gemessen wurde. Es zeigte sich eine Gleichwertigkeit der drei untersuchten Strategien bei dieser Ausgangssituation.

Nun wurden die Zyklen für Interrupt 1 und Interrupt 2 reduziert, bis eine Rechnerüberlastung bei der Strategie "FP" auftrat. Diese Situation ist in dem Protokoll in Bild 5.5 belegt (die Rechnerüberlastung ist an der maximalen Reaktionszeit bzw. Ausführungszeit für Task 2 zu erkennen: da das Maximum größer als die zugehörige Zykluszeit ist, muß ein Interrupt verloren gegangen sein).

Anschließend wurden die Zyklen für Interrupt 1 und Interrupt 2 weiter reduziert, bis auch bei der Strategie "KR" eine Rechnerüberlastung auftrat. Diese Situation ergab sich bei

Zyklus 1: 90 msec, Zyklus 2: 80 msec, Zyklus 3: 55 msec,  
Zyklus 4: 59 msec.

Eine Restzeitreduktion bei den Tasks 1 und 2 ergab jedoch einen fehlerfreien Ablauf (siehe Bild 5.6 und Bild 5.7).

. 0800

INT. ANZAHL: 4  
 VORWAHL INT. ABSTRENDE(USEC): MIN, MAX  
 INT. 1: 100000,  
 INT. 2: 090000,  
 INT. 3: 055000,  
 INT. 4: 059000,

INT. ABSTRENDE(USEC):  
 INT. 1: 100000 , 100000  
 INT. 2: 090072 , 090072  
 INT. 3: 055044 , 055044  
 INT. 4: 059214 , 059214  
 ANZ. INT. 1 VOR MESSPHASE: 3  
 ZUFALLSWORT:  
 MESSWERTE > 127998 USEC?: N  
 AENDERN?: N  
 INT. AUSGABE !  
 AUSWERTUNG?: J

ZEITEN/USEC	MIN	MITTELWERT	MAX
REAK. ZEIT INT. 1	000949	009898	034527
AUSF. ZEIT INT. 1	036585	054160	083375
REAK. ZEIT INT. 2	000947	052886	<u>118650</u>
AUSF. ZEIT INT. 2	040630	057875	<u>096875</u>
REAK. ZEIT INT. 3	000917	002003	011492
AUSF. ZEIT INT. 3	015871	016962	018638
REAK. ZEIT INT. 4	000945	003880	018832
AUSF. ZEIT INT. 4	008470	009216	011166

MESSWERTE DRUCKEN: N

Bild 5.5: Versuchsreihe 1 - Rechnerüberlastung bei Strategie "FP"

G800

INT. ANZAHL: 4  
 VORWAHL INT. ABSTRENDE(USEC): MIN, MAX  
 INT. 1: 090000,  
 INT. 2: 080000,  
 INT. 3: 055000,  
 INT. 4: 059000,

INT. ABSTRENDE(USEC):  
 INT. 1: 090072 , 090072  
 INT. 2: 080064 , 080064  
 INT. 3: 055044 , 055044  
 INT. 4: 059214 , 059214  
 ANZ INT. 1 VOR MESSPHASE: 3  
 ZUFALLSWORT:  
 MESSWERTE > 127998 USEC?: N  
 AENDERN?: N  
 INT. AUSGABE !  
 AUSWERTUNG?: J

ZEITEN/USEC	MIN	MITTELWERT	MAX
PEAK ZEIT INT. 1	000949	009988	034169
AUSF. ZEIT INT. 1	036738	047628	078142
PEAK ZEIT INT. 2	000871	027183	<u>109511</u>
AUSF. ZEIT INT. 2	038529	048539	079339
PEAK ZEIT INT. 3	000945	002507	017638
AUSF. ZEIT INT. 3	015873	018210	025386
PEAK ZEIT INT. 4	000945	004814	021060
AUSF. ZEIT INT. 4	008470	009822	017470

MESSWERTE DRUCKEN: N

Bild 5.6: Versuchsreihe 1 - Rechnerüberlastung bei Strategie "KR"

GG00

INT. ANZAHL: 4  
 VORWAHL INT. ABSTRENDE(USEC): MIN, MAX  
 INT. 1: 090000,  
 INT. 2: 080000,  
 INT. 3: 055000,  
 INT. 4: 059000,

INT. ABSTRENDE(USEC):  
 INT. 1: 090072 , 090072  
 INT. 2: 080064 , 080064  
 INT. 3: 055044 , 055044  
 INT. 4: 059214 , 059214  
 ANZ. INT. 1 VOR MESSPHASE: 3  
 ZUFALLSWORT:  
 MESSWERTE > 127998 USEC?: N  
 RENDERN?: N  
 INT. AUSGABE !  
 AUSWERTUNG?: J

ZEITEN/USEC	MIN	MITTELWERT	MAX
REAK. ZEIT INT. 1	000949	008500	029542
AUSF. ZEIT INT. 1	037355	048759	064496
REAK. ZEIT INT. 2	000947	004359	024203
AUSF. ZEIT INT. 2	039464	046906	060230
REAK. ZEIT INT. 3	000878	002937	012957
AUSF. ZEIT INT. 3	015871	018539	026175
REAK. ZEIT INT. 4	000945	006023	024923
AUSF. ZEIT INT. 4	008470	009863	016113

MESSWERTE DRUCKEN: N

Bild 5.7: Versuchsreihe 1 - keine Rechnerüberlastung bei Strategie "KR mit Restzeitreduktion"

Die Strategie "KR mit Restzeitreduktion" führte schließlich erst bei dem folgenden Interruptschema zu einer Rechnerüberlastung:

Zyklus 1: 60 msec, Zyklus 2: 60 msec, Zyklus 3: 55 msec,  
 Zyklus 4: 59 msec.

### 5.3.3. Dritte Versuchsreihe

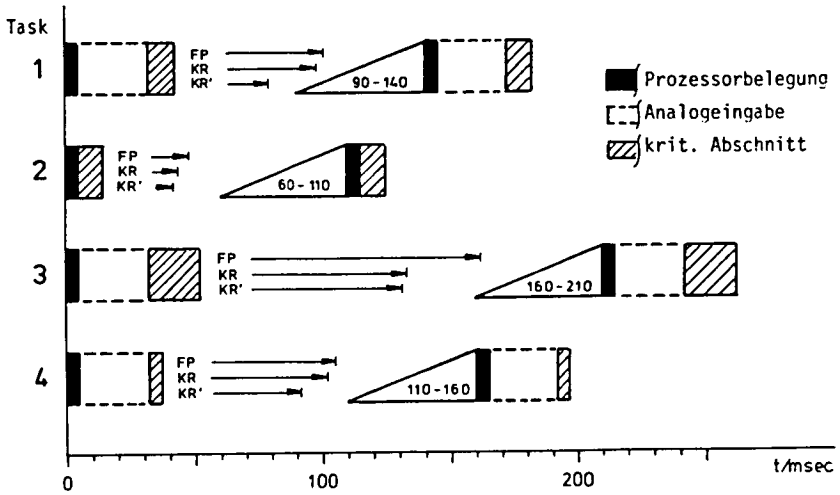
In der dritten Serie wurden statt Wartezeiten Analogeingaben vorgesehen. Diese erfolgten über einen Kanal, sodaß die Tasks um den Zugriff konkurrieren mußten. Die Interruptabstände wurden stochastischen Schwankungen unterworfen, die bei allen vier Tasks gleich groß waren. Die zeitlichen Abläufe der Antworttasks sind in Bild 5.8 dargestellt.

Task	1.Prozessorbelegung	Analog- eingabe	2.Prozessorbelegung (krit. Abschnitt)
1	5 msec	ja	10 msec
2	5 msec	nein	10 msec
3	5 msec	ja	20 msec
4	5 msec	ja	5 msec

Bild 5.8: Ablaufschema der Antworttasks bei der 3.Versuchsreihe

In den folgenden Diagrammen ist die mögliche zeitliche Verschiebung zweier aufeinanderfolgender Abläufe durch Dreiecke dargestellt: Der Abstand zwischen Ordinate und der linken unteren Ecke eines Dreieckes kennzeichnet die kleinstmögliche Zeitdifferenz zweier aufeinanderfolgender Starts der betreffenden Task. Analog dazu ist die größtmögliche Zeitdifferenz durch den Abstand zwischen Ordinate und der rechten senkrechten Seite repräsentiert. Durch die Pfeile werden die ermittelten maximalen Antwortzeiten angezeigt.

In Bild 5.9 ist die Situation gezeigt, bei der die Strategie "FP" zu einer stärkeren Rechnerbelastung als die Strategie "KR" führte, und in Bild 5.10 die Konstellation, bei der nur mehr die Strategie "KR mit Restzeitreduktion" einen einwandfreien Ablauf gewährleistete, wogegen die Strategie "KR" zu einer leichten "Überlastung" und die



FP...Strategie "FP", KR...Strategie "KR", KR'..."KR" mit Restzeitred.

Bild 5.9: Versuchsreihe 3 - Tasks mit Analogeingaben

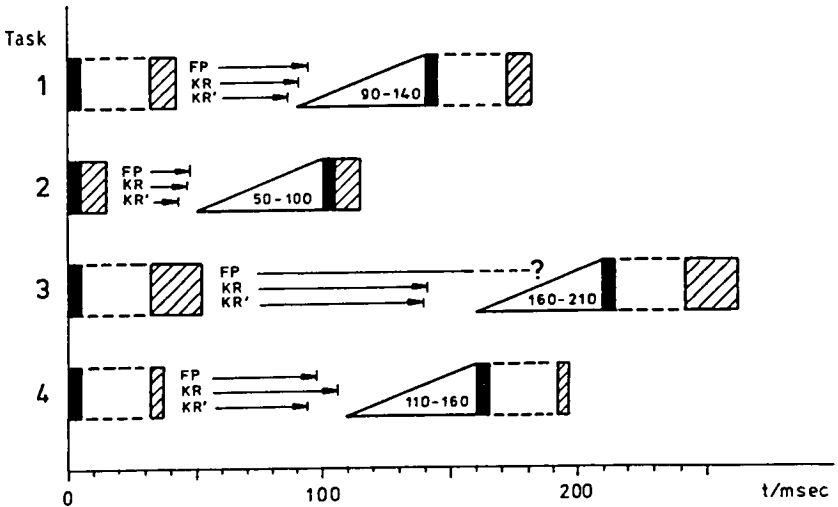


Bild 5.10: Versuchsreihe 3 - Rechnerüberlastung bei Strategie "FP"

Strategie "FP" sogar zum Verlust eines Interrupts führte.

An den Versuchsreihen 1 und 3 ist der maßgebliche Nachteil der Strategie "FP" gegenüber restzeitgesteuerten Strategien bereits deutlich zu erkennen, nämlich die Bevorzugung höherpriorer Tasks in jedem Fall, also auch dann, wenn für eine noch nicht beendete niedrigpriorer Task die Zeitschranke schon sehr nahe gerückt ist. Die Folge ist dann meist eine Verletzung dieser Zeitschranken durch nicht so dringliche Tasks. Für die wichtigsten Tasks ist die Einhaltung der vorgegebenen Zeitschranken durch die Strategie "FP" üblicherweise mindestens in ebendem Maße zu erzielen, wie mit restzeitgesteuerten Strategien.

#### 5.4. Kritik der Versuchsergebnisse

Der Einsatz des Prozeßrechnertestgerätes zum Vergleich von Taskverwaltungsstrategien erlaubte eine Untersuchung von wesentlich mehr verschiedenen exemplarischen Taskabläufen und Interruptschemata, als es mit einem Modellprozeß möglich gewesen wäre. Die dennoch geringe Anzahl der ausgewählten Fälle erlaubte es jedoch nicht, aus den gewonnenen Meßwerten statistisch untermauerte Schlüsse zu ziehen. Aus diesem Grund konnten die erzielten Ergebnisse nur als Bestätigung einer Tendenz insofern aufgefaßt werden, als trotz mehr oder weniger willkürlicher Auswahl der Parameter keine einzige Situation auftrat, die den Annahmen widersprach.

Es kann danach zumindest die Feststellung getroffen werden, daß sich im praktischen Versuch die Zweckmäßigkeit herausgestellt hat, für Prozeßrechner Betriebssysteme zu implementieren, die eine Umschaltung zwischen prioritäts- und restzeitgesteuerten Taskverwaltungsstrategien abhängig vom jeweiligen Anwendungsfall ermöglichen. Darüberhinaus scheint eine dynamische Modifizierbarkeit der Zeitschranken auf Sprachebene sinnvoll zu sein, insbesondere als dies kaum zusätzlichen Implementationsaufwand erfordert.

Eine Rechtfertigung des relativ hohen Aufwandes für die Realisierung des Konzepts der "kritischen Prioritäten" läßt sich aus den Versuchsergebnissen allein jedoch nicht ableiten.

### Zusammenfassung

In der vorliegenden Arbeit wurden die Zeitprobleme untersucht und klassifiziert, die beim Einsatz von Datenverarbeitungsanlagen im Bereich der Prozeßautomatisierung auftreten. Es wurde angedeutet, mit welchen Softwaremaßnahmen die verschiedenen zeitlichen Anforderungen behandelt werden können. Dabei zeigte sich, daß konventionelle Techniken für die Ablauforganisation von parallelen Aktivitäten - z.B. die Abarbeitung nach fest vorgegebenen Prioritäten - in einem großen Teil der Fälle neueren Strategien, die Zeitschranken als Entscheidungskriterien heranziehen, unterlegen sind.

Über eine Beschreibung verschiedener Strategien wurde demonstriert, daß unter gewissen einfachen Randbedingungen diese zeitschranken-gesteuerten Mechanismen ein formal optimales Verhalten aufweisen.

Daß auch in komplexeren Anwendungsfällen ihre Überlegenheit angenommen werden kann, und daß über zusätzliche Maßnahmen noch weitere Verbesserungen zu erzielen sind, wurde an Hand einer Modellimplementation verschiedener Strategien und deren Vergleich am praktischen Versuch gezeigt. Dabei stellte sich heraus, daß der Implementationsaufwand für die Strategie "nach der kürzesten Restzeit" mit dem für konventionelle Prioritätszahlen vergleichbar ist.

Darüberhinaus bedeutete es nur einen vernachlässigbaren Zusatzaufwand, den Betriebssystemkern derart auszulegen, daß eine einfache Umschaltung zwischen verschiedenen Strategien - je nach Anwendungsfall möglich wurde.

Auf Grund dieser Erkenntnisse scheint es erstrebenswert zu sein, in künftigen Betriebssystemen für Prozeßrechner derartige Vorkehrungen zu treffen, um im realen Einsatz weitergehende Vergleiche der betreffenden Strategien vornehmen zu können. Es wäre zu untersuchen, inwieweit dies auch durch Modifikationen in bereits existierenden Systemen geschehen könnte.



## ANHANG I

### ABLEITUNGEN ZU DEN KAPITELN 3.1.6. und 3.1.7.

#### I.1 Definitionen

Eine Menge von (zu Beginn des Beobachtungszeitraumes aktiven) Tasks, zu der während des gesamten Beobachtungszeitraumes keine neuen Tasks hinzukommen (durch Aktivierung o.ä.), wird als begrenzt bezeichnet.

Im Gegensatz dazu ist eine allgemeine oder nicht begrenzte Taskmenge dadurch charakterisiert, daß neue Tasks hinzukommen können.

Eine Menge von Tasks, deren Indizes der Regel

$$i < j \rightarrow r_i(t) \leq r_j(t) \quad \forall i, j : T_i \in \mathcal{M}(t), T_j \in \mathcal{M}(t)$$

genügen, wird als ausgerichtet bezeichnet, d.h. die Elemente der Menge sind nach aufsteigenden Restzeiten geordnet. Natürlich kann jede Taskmenge in eine ausgerichtete Taskmenge übergeführt werden.

Eine zeitgerechte Betriebsmittelbelegungssequenz liegt dann vor, wenn keine Zeitschranke verletzt wird.

Eine Taskmenge ist zeitgerecht verarbeitbar, wenn es eine zeitgerechte Betriebsmittelbelegungssequenz gibt.

Eine Strategie ist (unter gegebenen Randbedingungen) zeitgerecht, wenn sie für jede zeitgerecht verarbeitbare Taskmenge eine zeitgerechte Betriebsmittelbelegungssequenz liefert.

#### I.2 Ableitungen zum Kapitel 3.1.6.

Bei den folgenden Ableitungen ist angenommen, daß die betreffenden Tasks ausschließlich um einen Prozessor konkurrieren und daß keine gegenseitige Synchronisation erfolgt.

#### Satz 1

Eine begrenzte ausgerichtete Taskmenge  $\mathcal{M}(t)$  mit  $|\mathcal{M}(t)| = n$  ist dann und nur dann zeitgerecht verarbeitbar, wenn gilt

$$r_k(t) \geq \sum_{i=1}^k l_i(t) \quad \forall k : 1 \leq k \leq n \wedge T_k \in \mathcal{L}(t) \quad (1)$$

# Beweis

Siehe [HEN75, Seite 37]

## Satz 2

Wird aus einer zum Zeitpunkt  $t$  zeitgerecht verarbeitbaren begrenzten ausgerichteten Taskmenge  $\mathcal{M}(t)$  im Intervall  $[t, t + \Delta t)$  einer Task  $T_j \in \mathcal{G}(t)$  der Prozessor zugeteilt, so entsteht wieder eine zeitgerecht verarbeitbare Taskmenge, wenn gilt

$$S_j(t) + \Delta t \leq \min \{r_k(t) \mid T_k \in \mathcal{G}(t)\} \quad (2)$$

## Beweis

1. Aus

$$S_j(t) = r_j(t) - l_j(t)$$

und (2) folgt  $r_k(t) \geq r_j(t) - l_j(t) + \Delta t \quad \forall k : T_k \in \mathcal{G}(t)$

Wegen zeitgerechter Verarbeitbarkeit zu Zeitpunkt  $t$  gilt mit (1)

$$r_j(t) \geq \sum_{i=1}^j l_i(t)$$

Daraus folgt

$$r_k(t) \geq \sum_{i=1}^j l_i(t) - l_j(t) + \Delta t$$

$$r_k(t) \geq \sum_{i=1}^{j-1} l_i(t) + \Delta t$$

Mit

$$l_i(t + \Delta t) = l_i(t) \quad \forall i : i \neq j$$

und

$$r_k(t + \Delta t) = r_k(t) - \Delta t \quad \forall k : T_k \in \mathcal{G}(t)$$

gilt

$$r_k(t + \Delta t) + \Delta t \geq \sum_{i=1}^{j-1} l_i(t + \Delta t) + \Delta t$$

$$r_k(t + \Delta t) \geq \sum_{i=1}^k l_i(t + \Delta t) \quad \forall k : 1 \leq k < j \quad (3)$$

2. Mit

$$l_i(t + \Delta t) = l_i(t) \quad \forall i : i \neq j$$

und

$$l_j(t + \Delta t) = l_j(t) - \Delta t$$

sowie

$$r_k(t + \Delta t) = r_k(t) - \Delta t \quad \forall k : T_k \in \mathcal{G}(t)$$

gilt

$$r_k(t + \Delta t) \geq \sum_{i=1}^k l_i(t + \Delta t) \quad \forall k : j \leq k \leq n$$

Damit und mit (3) gilt schließlich

$$r_k(t + \Delta t) \geq \sum_{i=1}^k l_i(t + \Delta t) \quad \forall k : 1 \leq k \leq n$$

q.e.d.

$$\wedge T_k \in \mathcal{G}(t)$$

### Satz 3

Die Strategien "KR" und "KS" erfüllen beide die Bedingung (2)

#### Beweis

##### 1. Strategie "KR"

Bei der Strategie "KR" wird aus einer ausgerichteten Taskmenge  $\mathcal{M}(t)$  der Task  $T_i \in \mathcal{L}(t)$  der Prozessor zugeteilt.

Es gilt

$$r_i(t) = \min \{ r_k(t) \mid T_k \in \mathcal{L}(t) \}$$

Mit  $s_i(t) = r_i(t) - l_i(t)$

gilt  $s_i(t) + l_i(t) \leq \min \{ r_k(t) \mid T_k \in \mathcal{L}(t) \}$

q.e.d.

##### 2. Strategie "KS"

Bei der Strategie "KS" wird im Zeitintervall  $[t, t + \Delta t)$  mit  $\Delta t \rightarrow 0$  der Task  $T_j \in \mathcal{L}(t)$  der Prozessor zugeteilt, für die gilt

$$s_j(t) = \min \{ s_k(t) \mid T_k \in \mathcal{L}(t) \}$$

Wegen

$$s_k(t) = r_k(t) - l_k(t)$$

gilt

$$s_j(t) + l_k(t) \leq r_k(t) \quad \forall k : 1 \leq k \leq n$$

Wegen

$$\Delta t \rightarrow 0$$

gilt

$$\min \{ l_k(t) \mid 1 \leq k \leq n \} \geq \Delta t$$

und damit (2)

q.e.d.

### Satz 4

Eine Strategie, die die Bedingung (2) erfüllt, ist auch für nicht begrenzte (allgemeine) Taskmengen zeitgerecht

#### Beweis

Für eine zeitgerecht verarbeitbare nicht begrenzte Taskmenge  $\mathcal{M}(t_*)$  liege eine zeitgerechte Prozessorbelegungssequenz vor. Bis zum Zeitpunkt  $t_1$  sei die Bedingung (2) erfüllt, wogegen im Intervall  $[t_1, t_1 + \Delta t)$  die Bedingung (2) nicht erfüllt sei, d.h.

$$\exists T_j : S_j(t_1) + \Delta t > \min \{ r_k(t_1) \mid T_k \in \mathcal{G}(t_1) \}$$

Dieser Task  $T_j$  sei im Intervall  $[t_1, t_1 + \Delta t)$  der Prozessor zugeteilt.

Es gilt  $\exists T_i : S_i(t_1) + \Delta t \leq \min \{ r_k(t_1) \mid T_k \in \mathcal{G}(t_1) \}$

Daraus folgt  $r_j(t_1) - l_j(t_1) > r_i(t_1) - l_i(t_1)$

$$t_{zj} - l_j(t_1) > t_{zi} - l_i(t_1)$$

Wegen  $l_j(t_1) \geq \Delta t$

gilt  $t_{zj} - \Delta t > t_{zi} - l_i(t_1)$

Der Zeitpunkt  $t_{zi} - l_i(t_1)$  ist der spätestmögliche, ab dem der Task  $T_i$  der Prozessor zugeteilt sein muß, damit keine Verletzung von  $t_{zi}$  vorliegt. Daher sagt die Ungleichung (4) aus, daß vor dem Zeitpunkt  $t_{zj}$  der Task  $T_i$  mindestens für  $\Delta t$  der Prozessor zugeteilt sein muß.

Als Folge davon kann ein Anteil  $\Delta t$  dieser Zuteilung mit der Prozessorzuteilung für  $T_j$  im Intervall  $[t_1, t_1 + \Delta t)$  vertauscht werden, ohne daß deshalb eine Verletzung der Zeitschranken erfolgt.

Durch diese Vertauschung ist aber nun die Bedingung (2) bis zum Zeitpunkt  $t_1 + \Delta t$  erfüllt.

Dieser Schritt kann nun für das Intervall  $[t_1 + \Delta t, t_1 + 2\Delta t)$  wiederholt werden usw.

q.e.d.

#### Satz 5

Eine Strategie, die zu jedem Zeitpunkt  $t$  für alle Tasks  $T_i \in \mathcal{G}(t)$  die Restzeiten  $r_i(t)$  um einen beliebigen Anteil der jeweils noch bevorstehenden Laufzeiten  $\varepsilon_i \cdot l_i(t)$  ( $0 \leq \varepsilon_i < 1$ ) reduziert, und die dann jener Task mit der so entstandenen kleinsten "reduzierten Restzeit"  $r_i(t) - \varepsilon_i \cdot l_i(t)$  den Prozessor maximal für die Zeitdauer  $(1 - \varepsilon_i) \cdot l_i(t)$  zuteilt, ist für allgemeine Taskmengen zeitgerecht

#### Beweis

Es sei definiert  $\bar{r}_i(t) = r_i(t) - \varepsilon_i \cdot l_i(t)$

$$\exists j : \bar{r}_j(t) = \min \{ \bar{r}_k(t) \mid T_k \in \mathcal{G}(t) \}$$

Daraus folgt

$$r_j(t) - \varepsilon_j \cdot l_j(t) \leq \min \{ r_k(t) \mid T_k \in \mathcal{G}(t) \}$$

und schließlich 
$$S_j(t) + (1 - \varepsilon_i) \cdot l_i(t) \leq \min \{r_k(t) \mid T_k \in \mathcal{S}(t)\}$$

d.h. es gilt die Bedingung (2)

q.e.d.

Durch den Satz 5 ist eine Klasse von Strategien definiert, die unter den eingangs erwähnten Bedingungen zeitgerecht sind.

Für den Fall, daß alle  $\varepsilon_i$  gleich 0 sind, handelt es sich um die Strategie "KR".

Für den Fall, daß für alle  $i$  gilt  $\varepsilon_i \rightarrow 1$ , handelt es sich um die Strategie "KS".

### I.3 Ableitung zum Kapitel 3.1.7.

#### Satz 6

Eine begrenzte Taskmenge  $\mathcal{M}(t)$ , die zum Zeitpunkt  $t_0$  einerseits aus Tasks besteht, die nur den Prozessor anfordern (Gruppe a) und andererseits aus Tasks, die die Folge Prozessor - Kanalwerk anfordern (Gruppe b), wobei alle Kanalbelegungszeiten gleich lang sind, wird durch die Strategie "KR mit Restzeitreduktion" (unter Verwendung der Kanalbelegungszeit als Reduktion) mindestens ebenso gut bzw. besser abgearbeitet als durch die Strategie "KR", sofern durch letztere keine Zeitschranken verletzt werden.

#### Beweis

Es ist zu bemerken, daß bei "KR" die Kanalbelegungen genau in der Reihenfolge der aufsteigenden Restzeiten erfolgen, da die davorliegenden Prozessorbelegungsphasen diesem Gesetz ebenfalls genügen. Bei der Strategie "KR mit Restzeitreduktion" ist die Prioritätsbeziehung aller Tasks der Gruppe b, die die Kanalbelegung noch vor sich haben, untereinander genauso wie bei "KR", da von den betreffenden Restzeiten jeweils der gleiche Betrag (nämlich die Kanalbelegungszeit) abgezogen wird. Es ist aber möglich, daß Tasks der Gruppe b durch diese Restzeitreduktion wichtiger werden als solche der Gruppe a. Daher können im Vergleich zu "KR" Prozessor-

belegungen von Tasks der Gruppe a "nach hinten" und solche von Tasks der Gruppe b nach "vorne" verschoben werden. Jedenfalls kann dadurch keine Verschiebung von Kanalbelegungen nach "hinten" erfolgen.

Bei den erwähnten Verschiebungen der Prozessorbelegungen handelt es sich grundsätzlich nur um "Austauschschritte", wobei keine Zeitschranken von Tasks der Gruppe a verletzt werden können:

Sei  $l_{pb}$  die Prozessorbelegung einer Task  $T_b$  der Gruppe b und  $t_{Ea}$  der Zeitpunkt der Beendigung einer Task  $T_a$  der Gruppe a unter der Strategie "KR" sowie  $l_k$  die Kanalbelegungszeit, so erfolgt ein Austausch der Prozessorbelegungen nur dann, wenn gilt

$$t_{zb} - l_k < t_{za}$$

Angenommen, für  $T_a$  würde nun die Zeitschranke verletzt, dann müßte gelten

$$t_{za} < l_{pb} + t_{Ea} \quad (5)$$

da ja die Abarbeitung von  $T_a$  um  $l_{pb}$  verzögert wird. Wegen zeitge-rechter Abarbeitung von  $T_b$  unter "KR" gilt

$$t_{zb} \geq t_{Ea} + l_{pb} + l_k$$

Daraus folgt

$$t_{zb} - l_k \geq t_{Ea} + l_{pb}$$

$$\rightarrow t_{za} \geq t_{Ea} + l_{pb}$$

→ Widerspruch zu (5)

Anmerkung: Die Bedingung (5) für die Nichteinhaltung der Zeitschranke  $T_a$  gilt vorest nur, wenn die Task  $T_b$  vor dem Austausch unmittelbarer Nachfolger von  $T_a$  war. Man kann sich aber leicht davon überzeugen, daß bei einer Austauschreihenfolge derart, daß immer nur unmittelbar benachbarte Tasks berücksichtigt werden, schließlich alle Tasks erfaßt werden für die gilt

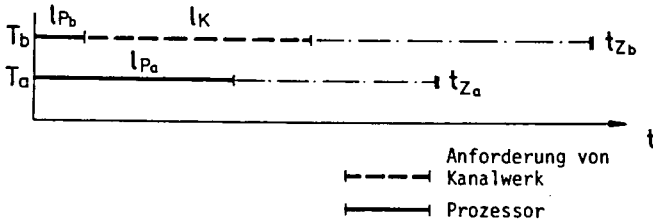
$$t_{zb} - l_k < t_{za}$$

Es gibt also:

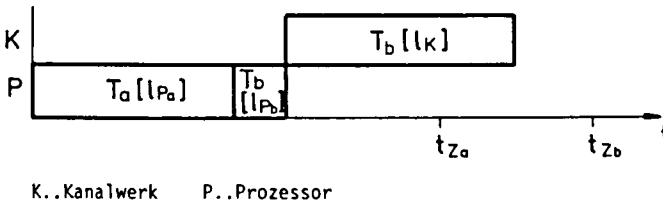
1. Keine Verschiebungen von Kanalbelegungen nach "hinten"
2. Verschiebungen von Prozessorbelegungen nach "hinten" nur

ohne Verletzung der Zeitschranken

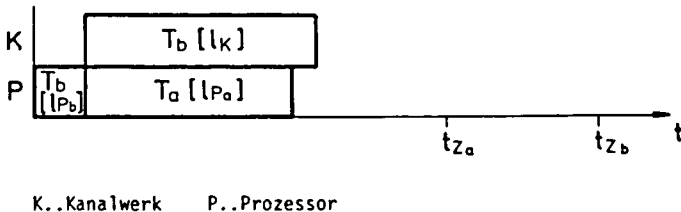
Daß es hingegen Verschiebungen von Kanalbelegungen nach "vorne" gibt, zeigt das folgende einfache Beispiel:



Bei Verarbeitung nach "KR" ergibt sich das folgende Belegungsschema:



Bei Verarbeitung nach der Strategie "KR mit Restzeitreduktion" können die Zeitschranken - wie man sieht - wesentlich kürzer vorgegeben sein:



In diesem Beispiel erfolgt ein Austausch von  $l_{Pa}$  und  $l_{Pb}$  sowie eine Vorwärtsverschiebung von  $l_K$ .

q.e.d.

## ANHANG II

### BESCHREIBUNG DER BAUSTEINE DES MODELLBETRIEBSSYSTEMS

In den folgenden Abschnitten werden die vom Modellbetriebssystem benötigten Datenstrukturen sowie die Verwaltungsbausteine beschrieben. Die Kenntnis der Programmiersprache PEARL wird vorausgesetzt.

#### II.1. Verwaltungsdaten

Ein großer Teil der für die Verwaltung der Rechenprozesse (Tasks), der Geräte, der Einplanungsbedingungen etc. notwendigen Datenstrukturen muß im Anwenderprogramm bereitgestellt werden. Dadurch wird erreicht, daß trotz Fehlens einer Speicherverwaltung im Betriebssystem die Anzahl der zu verwaltenden Einheiten theoretisch unbegrenzt ist.

Die folgenden 7 Verwaltungsdatenstrukturen sind vorgesehen:

- Taskverwaltungsblöcke
- Bolt-Verwaltungsblöcke
- Semaphore-Verwaltungsblöcke
- Geräteverwaltungsblöcke
- Auftragsparameterblöcke
- Interruptliste
- Zeitliste

Für die Verkettung von Verwaltungsdaten zur Laufzeit werden 3 verschiedene Verfahren angewandt:

- Warteschlangen
- "geschlossene" Ketten
- "offene" Ketten

##### a. Warteschlangen

Warteschlangen sind doppelt verzeigert (jedes Element enthält je einen Zeiger auf "Vorgänger" bzw. "Nachfolger"), um ein schnelles Ausketten eines Elementes zu ermöglichen. Die Elemente sind nach einer Prioritätszahl geordnet. Der Warteschlangenkopf (Anfang der Warteschlange) enthält je einen Zeiger auf das erste bzw. letzte Element in der Warteschlange.



Warteschlangen werden zur Bearbeitung von Konkurrenzsituationen der Tasks um Betriebsmittel bzw. Synchronisationsvariable benutzt. Jede Task kann sich in höchstens einer Warteschlange befinden.

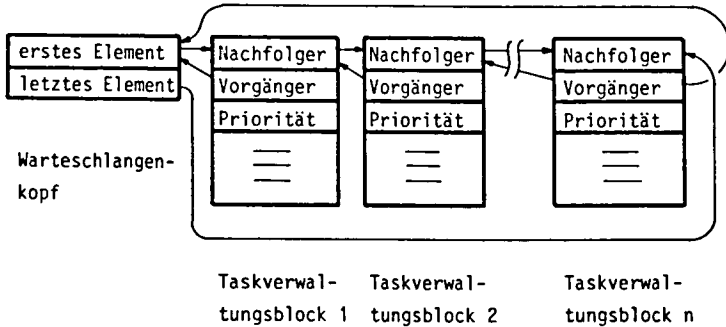


Bild II.1: Prinzip der Verkettung von Taskverwaltungsblöcken über Warteschlangen

Es existieren die folgenden Warteschlangen:

- Prozessorwarteschlange (PWS) verkettet alle lauffähigen Rechenprozesse. Der erste Rechenprozeß hat die Kontrolle über den Prozessor. Der Warteschlangenkopf liegt im Betriebssystem.
- Bolt-Warteschlange (BWS) verkettet alle auf die Freigabe der betreffenden Boltvariablen wartenden Tasks. Der Warteschlangenkopf ist Bestandteil des entsprechenden Bolt-Verwaltungsblockes.
- Semaphore-Warteschlange (SWS) verkettet alle auf die Freigabe der betreffenden Semaphorevariablen wartenden Tasks. Der Warteschlangenkopf ist Bestandteil des entsprechenden Semaphore-Verwaltungsblockes.

- Gerätewarteschlange (GWS)

verkettet alle Tasks, die sich für einen Transfer über das betreffende Gerät angemeldet haben, ausgenommen jene, deren Auftrag gerade bearbeitet wird. Der Warteschlangenkopf ist Bestandteil des entsprechenden Geräteverwaltungsblockes.

b. "geschlossene Ketten"

Geschlossene Ketten sind nur einfach verzeigert, wobei das letzte Element der Kette auf den Kettenanfang zeigt. Sie werden zur Verzeigerung von Einplanungsbedingungen ("Schedules") verwendet, die dem selben Ereignis zugeordnet sind.

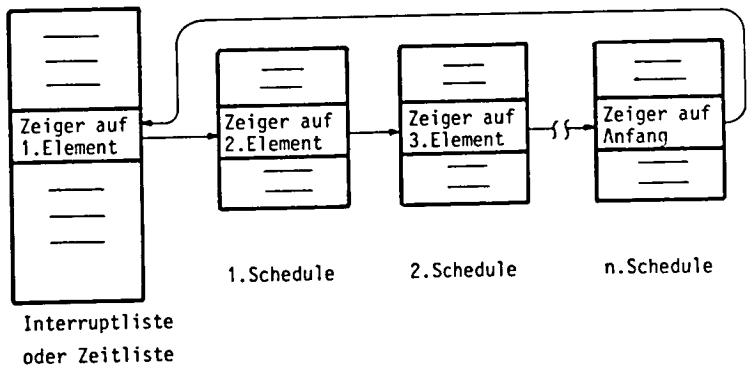


Bild II.2: Schedule-Kette (geschlossene Kette)

Der Kettenanfang ist durch eine Zelle der Interruptliste bzw. der Zeitliste repräsentiert und identifiziert das Ereignis, das zur Auswertung der betreffenden Einplanungsbedingungen führt. Die "geschlossene" Verkettung ist notwendig, da ein Ausketten eines Elementes möglich sein muß, ohne den Kettenanfang zu kennen. Die aus der einfachen Verzeigerung sich ergebende längere Bearbeitungszeit wurde hier in Kauf genommen und fällt dann nicht ins Gewicht, wenn die Wahrscheinlichkeit gering ist, daß sich mehrere Schedules auf das gleiche Ereignis beziehen.

### c. "offene Ketten"

Offene Ketten sind einfach verzeigert, wobei das letzte Element auf "nichts" zeigt. Sie sind dann sinnvoll, wenn keine Auskettungen erfolgen, bzw. wenn streng nach LIFO ("last-in-first-out", Kellertechnik) vorgegangen wird. Sie werden zur Verkettung der durch eine Task blockierten Bolt-Variablen verwendet. Hier erfolgt eine Einschränkung der Semantik der PEARL-Anweisungen RESERVE und FREE derart, daß jeweils diejenige Bolt-Variable zuerst freigegeben werden muß, die zuletzt reserviert wurde (Verschachtelung der kritischen Abschnitte):

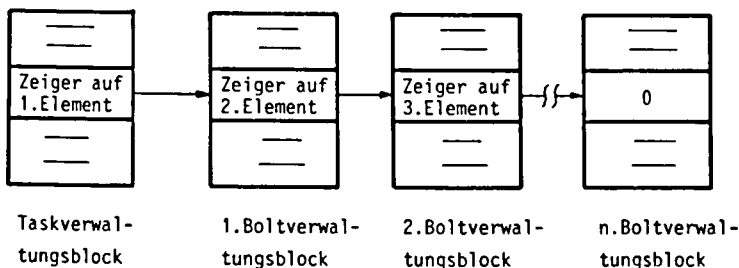


Bild II.3: Task-Bolt-Kette (TBK)

Offene Ketten werden außerdem für die (statische) Verzeigerung aller dem System bekannten Verwaltungsblöcke verwendet, um bei einem etwaigen Wiederanlauf Initialisierungen vornehmen zu können.

#### II.1.1. Taskverwaltungsblock (Prozeßkontrollblock)

In den Taskverwaltungsblöcken werden alle jene Informationen geführt, die den Zustand einer Task eindeutig beschreiben, sowie die (anlagen-abhängigen) Registerinhalte, die zur definierten Fortsetzung einer Task nach einer Unterbrechung notwendig sind.

In Bild II.4 sind alle für die Abläufe im Modellbetriebssystem notwendigen Informationen des Taskverwaltungsblockes beschrieben.

Für alle benötigten Taskverwaltungsblöcke muß im Anwenderprogramm Speicherplatz bereitgestellt werden. Dieser darf während der gesamten Lebensdauer des Systems nicht durch andere Daten überlagert werden. Die Taskverwaltungsblöcke können allerdings an beliebiger Stelle im (absolut durchadressierbaren) Speicher liegen.

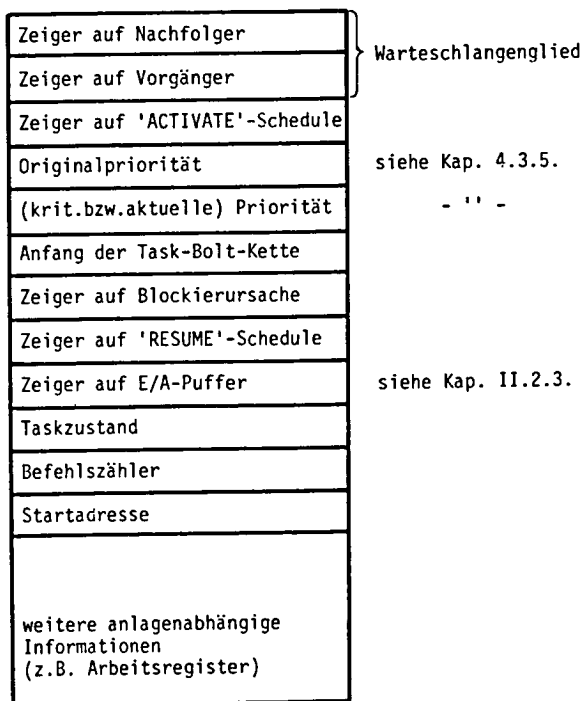


Bild II.4: Task-Verwaltungsblock

### II.1.2. Bolt-Verwaltungsblock

Für jede verwendete Bolt-Variable muß im Anwenderprogramm ein Bolt-Verwaltungsblock bereitgestellt werden, für den die Aussagen in den beiden letzten Sätzen von Kapitel II.1.1. analog gelten. Er enthält Informationen, die zur Repräsentation des in Kapitel 4.3.5. beschriebenen Bolt-Belegungsgraphen erforderlich sind.

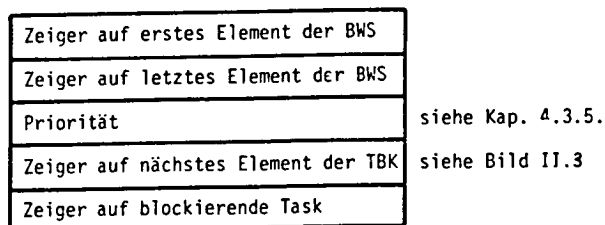


Bild II.5: Bolt-Verwaltungsblock

### II.1.3. Semaphor-Verwaltungsblock

Für jede verwendete Semaphor-Variable muß im Anwenderprogramm ein Semaphor-Verwaltungsblock bereitgestellt werden, für den die Aussagen in den beiden letzten Sätzen von Kapitel II.1.1. analog gelten. Er enthält den aktuellen Wert sowie den Anfangswert der Semaphore (letzteren für Initialisierungen), den Kopf der Semaphor-Warteschlange und einen Zeiger auf einen etwaigen 'RELEASE-Schedule'.

Zeiger auf erstes Element der SWS
Zeiger auf letztes Element der SWS
Zeiger auf 'RELEASE'-Schedule
Anfangswert
aktueller Wert

Bild II.6: Semaphor-Verwaltungsblock

### II.1.4. Geräteverwaltungsblock

Für jedes Gerät, das über die Ein/Ausgabeverwaltung bedient wird (siehe II.2.3.), ist im anlagenabhängigen Teil des Betriebssystems ein Geräteverwaltungsblock bereitzustellen. Er enthält einen Zeiger auf den Verwaltungsblock jenes Rechenprozesses, der das Gerät gerade belegt, sowie den Kopf der Warteschlange aller auf die Zuteilung des Gerätes wartenden Rechenprozesse (Gerätewarteschlange).

Darüberhinaus enthält er die Adresse einer geräteabhängigen Befehlsfolge, die beim Anstoß eines Transfers zu durchlaufen ist.

Zeiger auf erstes Element der GWS
Zeiger auf letztes Element der GWS
Zeiger auf Task, die Gerät belegt
Adr. der geräteabhängigen E/A-Routine

Bild II.7: Geräteverwaltungsblock

### II.1.5. Auftragsparameterblock

Die Schnittstelle zwischen Anwendertasks und Betriebssystem ist für alle Aufrufe einheitlich definiert. Sie ist durch den Verwaltungsbaustein AUFTRAGSÜBERNAHME repräsentiert, an den die Adresse eines Auftragsparameterblockes per Register übergeben wird. Dieser Auftragsparameterblock enthält die zur Ausführung der betreffenden Anweisung notwendigen Informationen - z.B. auch die Einplanungsbedingungen.

Die beiden folgenden Eintragungen sind für alle Aufrufe äquivalent:

- Zeiger auf Verwaltungsblock  
enthält die Adresse des Verwaltungsblockes der Task (der Boltvariablen, der Semaphore, des Gerätes), auf die sich der Aufruf bezieht. Diese Adresse kann 0 sein, wenn sich ein 'Tasking'-Statement auf die aufrufende Task bezieht.
- Zeiger auf aufzuführende Routine  
enthält die Adresse der für die Ausführung der betreffenden Anweisung zuständigen Betriebssystemroutine.

Die weiteren Informationen sind unterschiedlich, je nachdem, ob es sich um einen Auftrag an die Taskverwaltung oder an die Ein/Ausgabeverwaltung handelt.

Bei einem Auftrag an die Ein/Ausgabeverwaltung kommt nur noch die Adresse des E/A-Puffers dazu. Weitere geräteabhängige Steuerinformationen (z.B. Vorschubsteuerung für Drucker) sind unter dieser Pufferadresse zu führen und werden vom anlagenunabhängigen Teil des Betriebssystems nicht ausgewertet.

Adresse des Geräteverwaltungsblockes
Treiberadresse
Pufferadresse

siehe Kap.II.2.3.

Bild II.8: Auftragsparameterblock für Ein/Ausgabe

Für den Fall, daß mehrere Tasks das gleiche Gerät beauftragen, müssen alle Puffer verschieden sein (lokal zur Task). Ausgenommen davon sind natürlich Ein/Ausgabeanweisungen, die durch gegenseitigen Ausschluß explizit im Anwenderprogramm synchronisiert sind.

Der im Auftragsparameterblock angegebene Treiber ist bei Geräten, die "standardmäßig" bearbeitbar sind, identisch mit der in Kapitel II.2.3. beschriebenen E/A-Anstoßroutine. Andernfalls ist die Adresse einer gerätespezifischen Treiberroutine im anlagenabhängigen Teil des Betriebssystems anzugeben.

Bei einem Auftrag an die Taskverwaltung kommen die folgenden Informationen dazu:

- Typ des 'Schedules' (bzw. 0, wenn keine Einplanungsbedingung vorliegt)
- Priorität (falls diese zu ändern ist)
- Einplanungsbedingungen (nur bei Auftrag mit Schedule)
- Schedule-Ketten-Zeiger (zeigt auf 'nächsten' Schedule, wenn mehrere Schedules sich auf dasselbe Ereignis beziehen)

Zeiger auf Verwaltungsblock (Task,...)
Adresse der zuständigen Routine
Schedule-Ketten-Zeiger
Original-Schedule-Typ
aktueller Schedule-Typ
Priorität bzw. Zeitschranke
'WHEN'- Bedingung
'AFTER'- Bedingung
'ALL'- Bedingung

siehe Kap. II.2.4.

Bild II.9: Auftragsparameterblock für 'Tasking'- bzw. Synchronisationsanweisungen (Schedule)

Jene Informationen, die für den betreffenden Betriebssystemaufruf nicht relevant sind, können entfallen (Priorität, Einplanungsbedingungen). Falls der Auftrag sofort auszuführen ist (ohne Einplanungsbedingungen), kann der für den Auftragsparameterblock benötigte Speicherplatz nach Ausführung der Anweisung anderwärtig verwendet werden.

Sind jedoch Einplanungsbedingungen vorgesehen, so muß die betreffende Datenstruktur über die ganze Lebensdauer des Schedules für Betriebssystemmanipulationen zur Verfügung stehen. Insbesondere ist darauf zu achten, daß bei einem Auftrag, der innerhalb einer Prozedur eingeplant wird und der sich auf die aufrufende Task bezieht, der entsprechende Auftragsparameterblock n mal abgelegt wird, wenn die Prozedur von n Tasks aufgerufen wird.

#### II.1.6. Interruptliste

Die Unterbrechungsverwaltung bietet die Möglichkeit, mehrere Alarm-signale der Prozeßperipherie zu identifizieren. Jedem Alarm ist ein Listenplatz in der Interruptliste zugeordnet, der den Anfang einer möglichen Schedule-Kette darstellt. Falls für einen Alarm keine Reaktion eingeplant ist, zeigt der betreffende Zeiger auf sich selbst. Die Interruptliste ist Bestandteil des Betriebssystems.

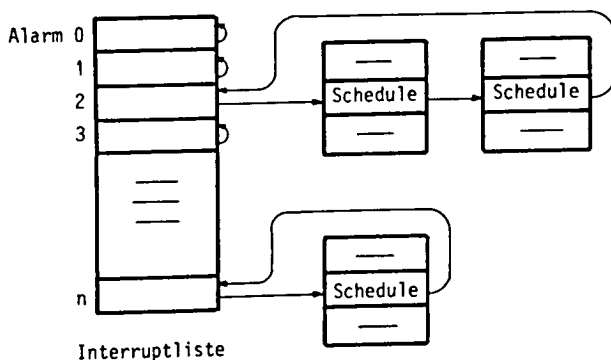


Bild II.10: Interruptliste mit möglichen Schedule-Einträgen



### II.1.7. Zeitliste

Der Aufbau der Zeitliste entspricht dem der Interruptliste. Es existiert ein Zeiger, der vom Zeittakt des Rechners weitergeschaltet wird und der die Zeitliste umlaufend bestreicht. Eine Aktion wird erforderlich, wenn dieser Zeiger auf einen Listenplatz zeigt, in den Schedules eingekettet sind.

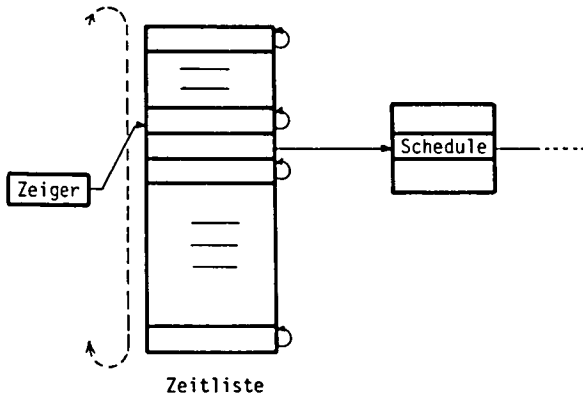


Bild II.11: Zeitliste

## II.2. Verwaltungsbausteine

### II.2.1. Prozessorverwaltung

Die Prozessorverwaltung ist der einfachste Baustein des Modellbetriebssystems. Sie wird grundsätzlich dann angestoßen, wenn nach der Bearbeitung eines beliebigen Aufrufes die Kontrolle über den Prozessor wieder an eine Anwendertask übergeben werden soll.

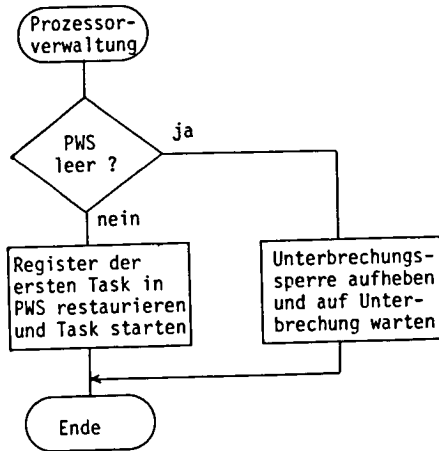


Bild II.12: Flußplan der Prozessorverwaltung

### II.2.2. Unterbrechungsverwaltung

Die Unterbrechungsverwaltung geht davon aus, daß die Unterbrechungssignale bereits durch die Hardware soweit identifiziert werden, daß

- für jedes E/A-Gerät
- für den Zeitgeber
- für das Sammelsignal "Alarm"

eine zugeordnete Speicheradresse angesprungen wird. Beim Alarmsignal erfolgt darüberhinaus ein "Multiplexen" dadurch, daß über ein Register die Nummer des verursachenden Alarmes übergeben wird. Falls die Zielmaschine diesen Mechanismus nicht zur Verfügung stellt - wie auch die im Rahmen dieser Arbeit verwendete Analge SIEMENS-306 - so müssen die entsprechenden Vorkehrungen im anlagenabhängigen Teil des Betriebssystems per Software getroffen werden.

#### II.2.2.1. Routinen für Zeitgeber und Alarm-Unterbrechung

Die Reaktionen auf Zeitgeber- bzw. Alarm-Unterbrechungen unterscheiden sich nur in der Auffindung des Zeitlisten- bzw. Interruptlistenplatzes, der den Anfang einer Schedule-Kette darstellt, falls für den betreffenden Zeitpunkt bzw. Alarm Einplanungen vorliegen.

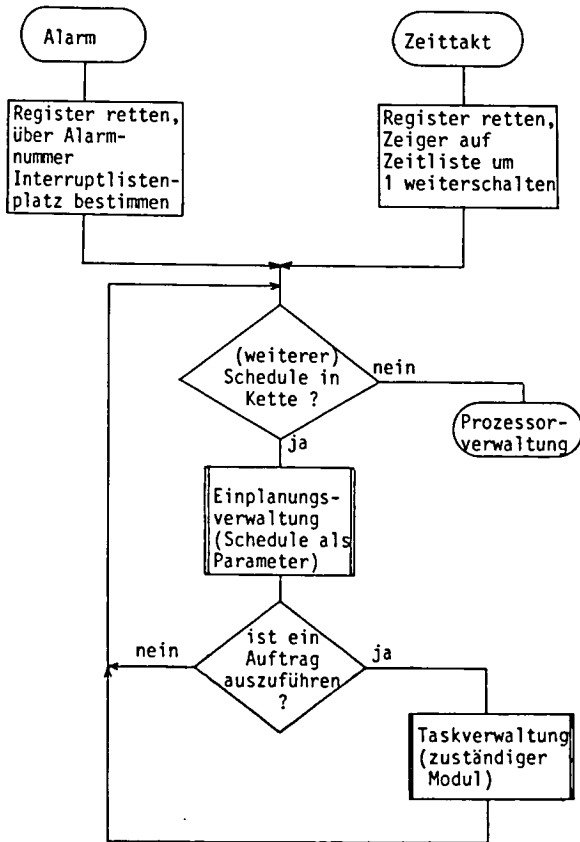


Bild II.13: Flußplan der Unterbrechungsverwaltung für Alarm und Zeittakt

#### II.2.2.2. Fertigmeldungen von Geräten

Das Modellbetriebssystem setzt voraus, daß Unterbrechungssignale von Geräten nur bei Beendigung des Transfers an die Unterbrechungsverwaltung gemeldet werden. Bezieht sich ein E/A-Aufruf auf einen ganzen Puffer, wobei das Gerät aber z.B. nach Übertragung jedes Zeichens ein Unterbrechungssignal liefert, so ist dieses im anlagenabhängigen Teil des Betriebssystems zu bearbeiten (z.B. mit

höherer Hardware-Priorität). Erst nach Übertragung des letzten Zeichens ist die Unterbrechungsverwaltung anzustoßen. Eine andere Lösung für diesen Anwendungsfall besteht darin, das Packen bzw. Entpacken von Puffern im Anwenderprogramm vorzunehmen (z.B. über geeignete Bibliotheksroutinen), und das Betriebssystem nur mit der Übertragung jeweils eines Zeichens zu beauftragen.

Da die Routinen zur Bearbeitung von E/A-Unterbrechungssignalen zum Teil geräteabhängig sind, ist in Bild II.14 ein Beispiel (für den Schnelldrucker der SIEMENS 306) angeführt. Geräteabhängige Befehlsgruppen sind in schraffierten Feldern dargestellt.

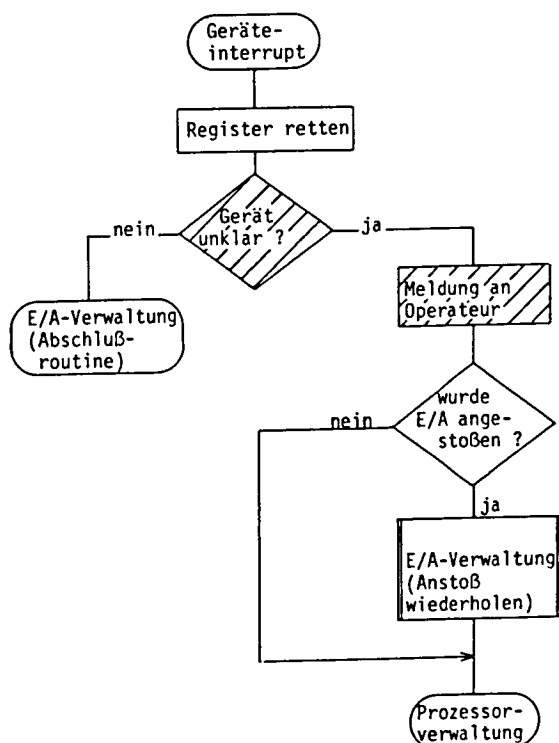


Bild II.14: Flußplan der Unterbrechungsverwaltung für Geräteinterrupt (Beispiel)

### II.2.3. Ein/Ausgabeverwaltung

Die Ein/Ausgabeverwaltung kann für alle Geräte benutzt werden, die durch den folgenden Mechanismus bedienbar sind: Der Anstoß eines Transfers erfolgt mit einer (geräteabhängigen) Maschinenbefehlsfolge und veranlaßt die Übertragung eines Datenpuffers beliebiger Länge vom/zum Gerät. Das Gerät meldet den Abschluß des Transfers mit einem Unterbrechungssignal an die Unterbrechungsverwaltung, die schließlich den Anstoß der E/A-Abschlußroutine bewirkt.

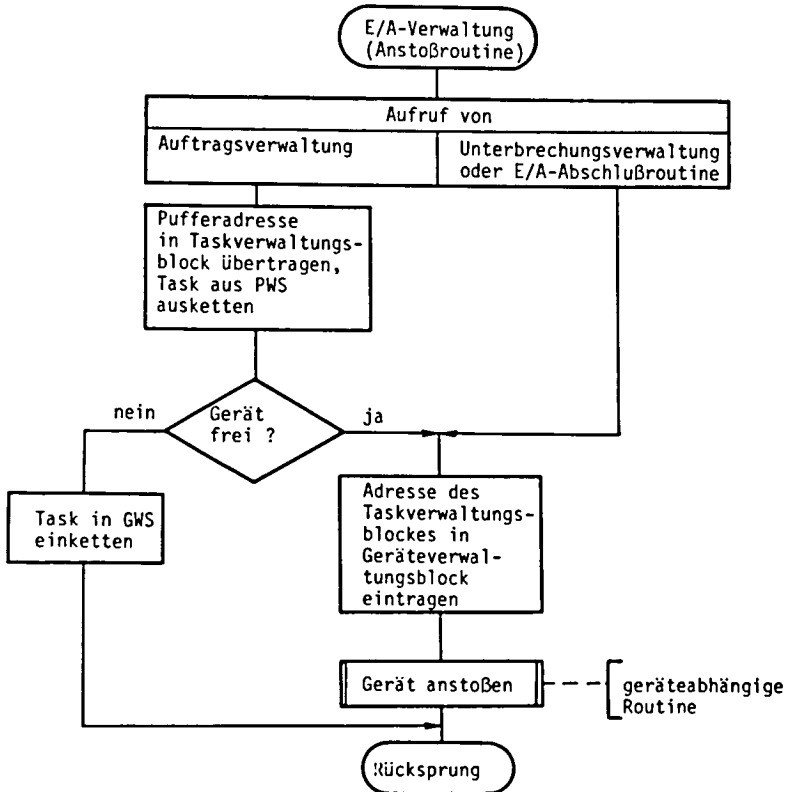


Bild II.15: E/A-Anstoßroutine der E/A-Verwaltung

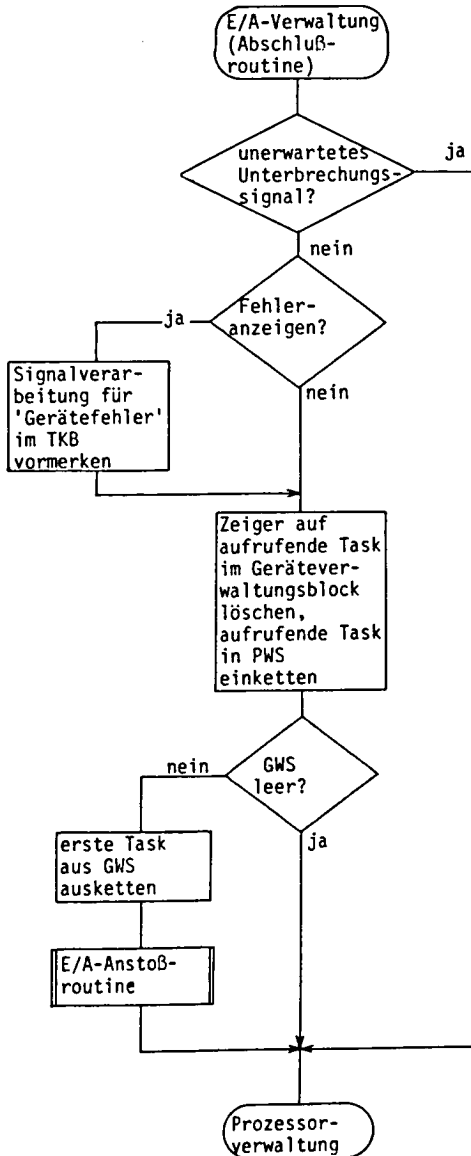


Bild II.16: E/A-Verwaltung (Abschlußroutine)

#### II.2.4. Einplanungsverwaltung

Die Einplanungsverwaltung wird von der Taskverwaltung beauftragt, die Ausführung bestimmter Anweisungen (ACTIVATE, RESUME, RELEASE) in Abhängigkeit von Alarmereignissen und/oder Zeitbedingungen vorzumerken. Dies geschieht durch Einketten des vom Rechenprozeß zur Verfügung gestellten Schedules in die Interruptliste bzw. Zeitliste. Sie hat außerdem die Aufgabe, zum Zeitpunkt des Eintretens eines mit einem Schedule verknüpften Ereignisses der aufrufenden Umgebung zurückzumelden, ob eine (und welche) Anweisung auszuführen ist, bzw. eine 'Weiterschaltung' (Umkettung) des Schedules vorzunehmen, falls die angegebenen Einplanungsbedingungen noch nicht vollständig erfüllt sind, oder eine zyklische Ausführung vorgesehen ist.

Zu diesem Zwecke wird im Schedule neben dessen 'Originaltyp' (siehe Kapitel 4.3.5.) eine sogenannte 'aktuelle' Typkennung geführt, die den jeweiligen Verarbeitungszustand kennzeichnet. Beim Absetzen eines Auftrages mit Einplanungsbedingungen seitens einer Anwendertask wird der aktuelle Typ mit dem Originaltyp vorbesetzt.

In Bild II.17 sind die verwendeten Typkennungen angeführt.

SCHEDULE	TYP
WHEN AFTER ALL	17
WHEN AFTER	16
WHEN ALL	13
WHEN	12
AFTER ALL	9
AFTER	8
ALL	5
abgelaufen	4

Bild II.17: Schedule-Typkennungen

Handelt es sich um einen mit einer 'WHEN'-Bedingung eingeleiteten Schedule, so erfolgt eine Subtraktion der Typkennung um 8, wodurch sich die Kennung der noch verbleibenden Bedingungskombination ergibt. Analog erfolgt bei einem durch eine 'AFTER'-Bedingung eingeleiteten Schedule eine Reduktion um 4.

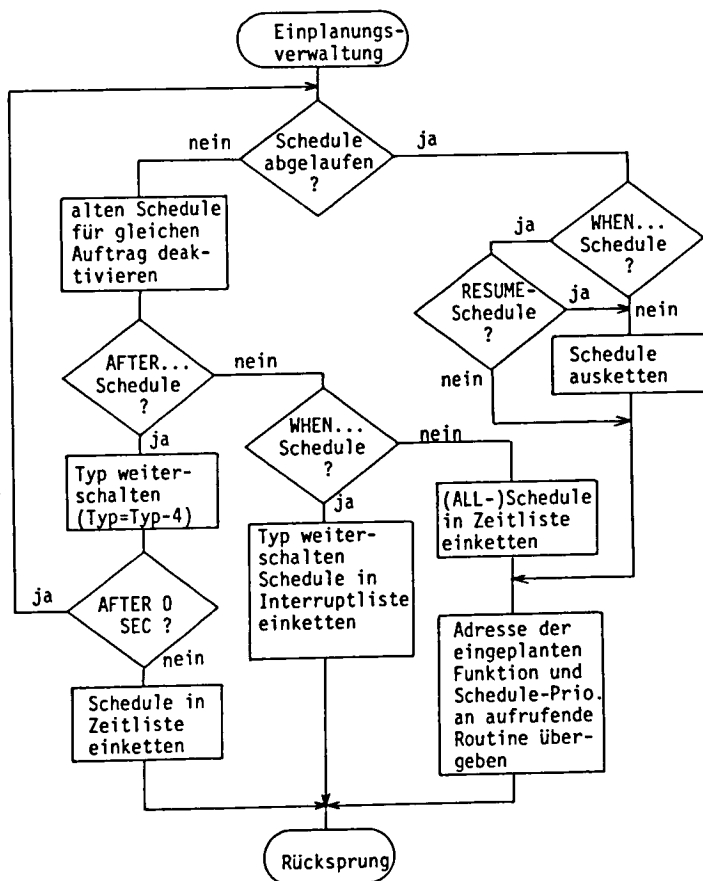


Bild II.18: Flußplan der Einplanungsverwaltung



### II.2.5. Taskverwaltung

Die Taskverwaltung wickelt alle Aufträge ab, die sich unmittelbar auf Tasks oder auf Synchronisationsvariable beziehen. Bei Angabe von Einplanungsbedingungen wird zwischenzeitlich die Einplanungsverwaltung eingeschaltet. Es existieren die folgenden 9 Moduln:

- |             |   |
|-------------|---|
| - ACTIVATE  | Starten einer Task                                    |
| - TERMINATE | Beenden einer Task                                    |
| - CONTINUE  | Änderung von Priorität bzw. Zeitschranke              |
| - RESUME    | Unterbrechung einer Task mit eingeplanter Fortsetzung |
| - PREVENT   | Löschen von Einplanungen                              |
| - REQUEST   | Belegen einer Semaphore                               |
| - RELEASE   | Freigeben einer Semaphore                             |
| - RESERVE   | Belegen einer Bolt-Variablen                          |
| - FREE      | Freigeben einer Bolt-Variablen                        |

#### II.2.5.1. Der Modul ACTIVATE

Der Modul ACTIVATE bearbeitet einen Startaufruf für eine Task, der mit Einplanungsbedingungen verknüpft sein kann. Wenn keine Priorität bzw. Antwortzeit angegeben ist, wird ersatzweise eine Minimalpriorität (bzw. maximale Antwortzeit) vorgesehen (siehe Abb. II.19).

#### II.2.5.2. Der Modul TERMINATE

Der Modul TERMINATE veranlaßt die Beendigung einer Task. Falls die zu beendende Task noch Boltvariable blockiert, wird zuerst eine Freigabe derselben veranlaßt. Bei einer 'Fremdterminierung' - also dann, wenn sich der Auftrag zur Beendigung nicht auf die aufrufende Task bezieht, wird die zu beendende Task dazu veranlaßt, sich selbst abzubrechen, indem ihr Befehlszähler auf einen entsprechenden Aufruf gesetzt wird. Zusätzlich ist es eventuell noch erforderlich, sie aus dem Zustand 'blockiert' in den Zustand 'bereit' zu versetzen.

Diese etwas umständlich erscheinende Vorgangsweise erleichtert das Freigeben von eventuell belegten Boltvariablen. Sie hat jedoch zur Folge, daß die gewünschte Zustandsänderung nach Durchlaufen des Aufrufs 'TERMINATE' im allgemeinen noch nicht erfolgt ist.

Die in Bild II.20 durch eine strichlierte Linie umschlossene

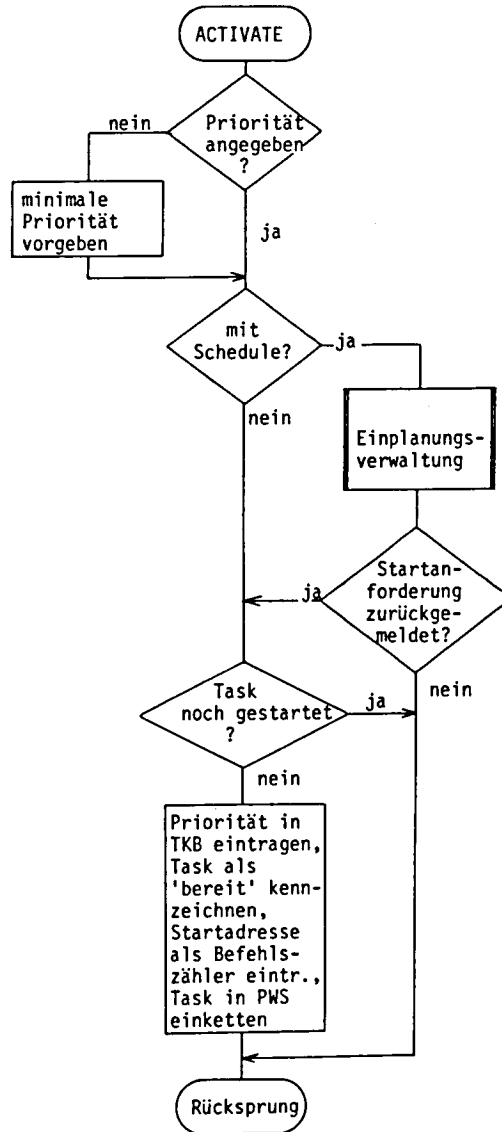


Bild II.19: Taskverwaltung - Modul ACTIVATE

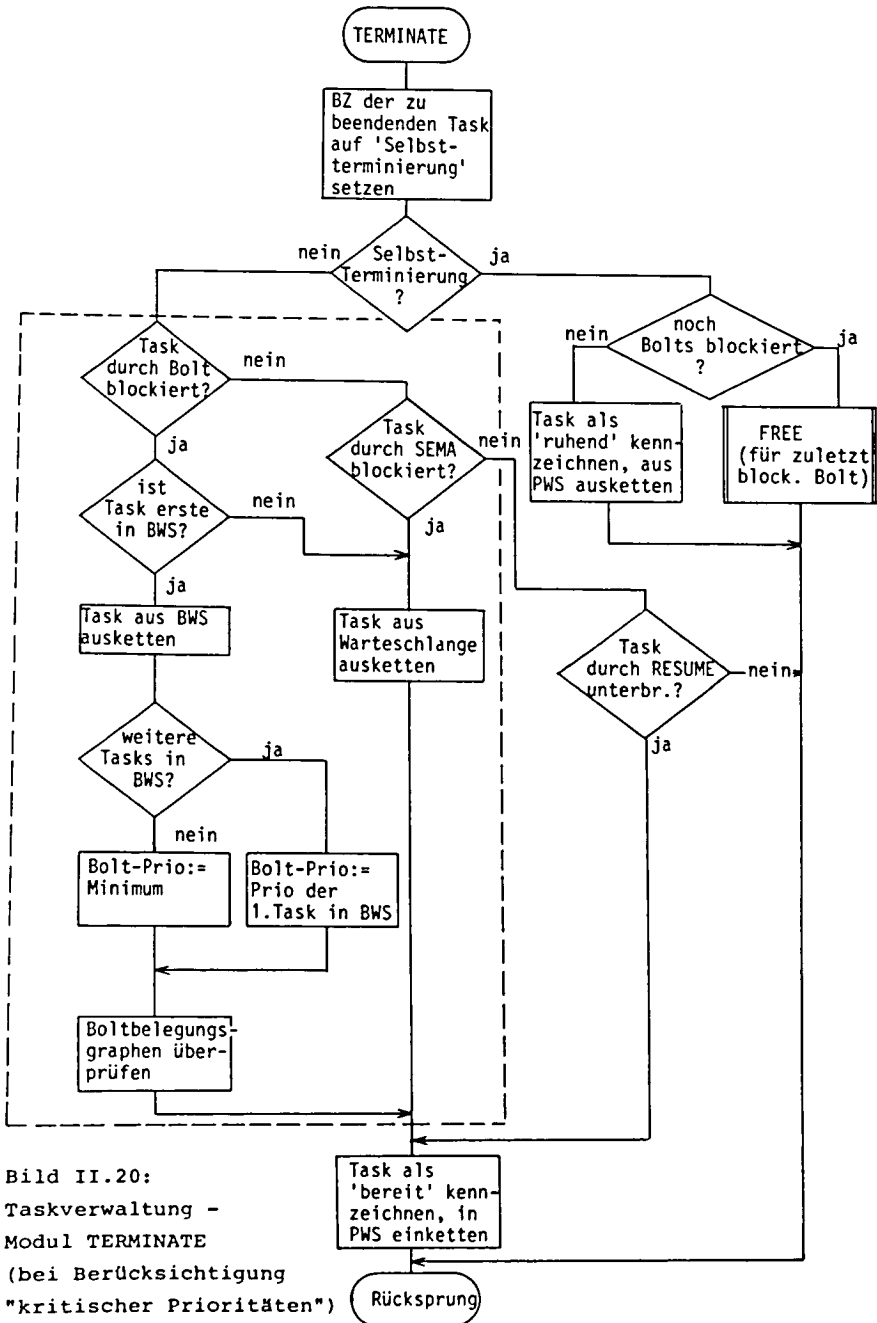


Bild II.20:  
Taskverwaltung -  
Modul TERMINATE  
(bei Berücksichtigung  
"kritischer Prioritäten")

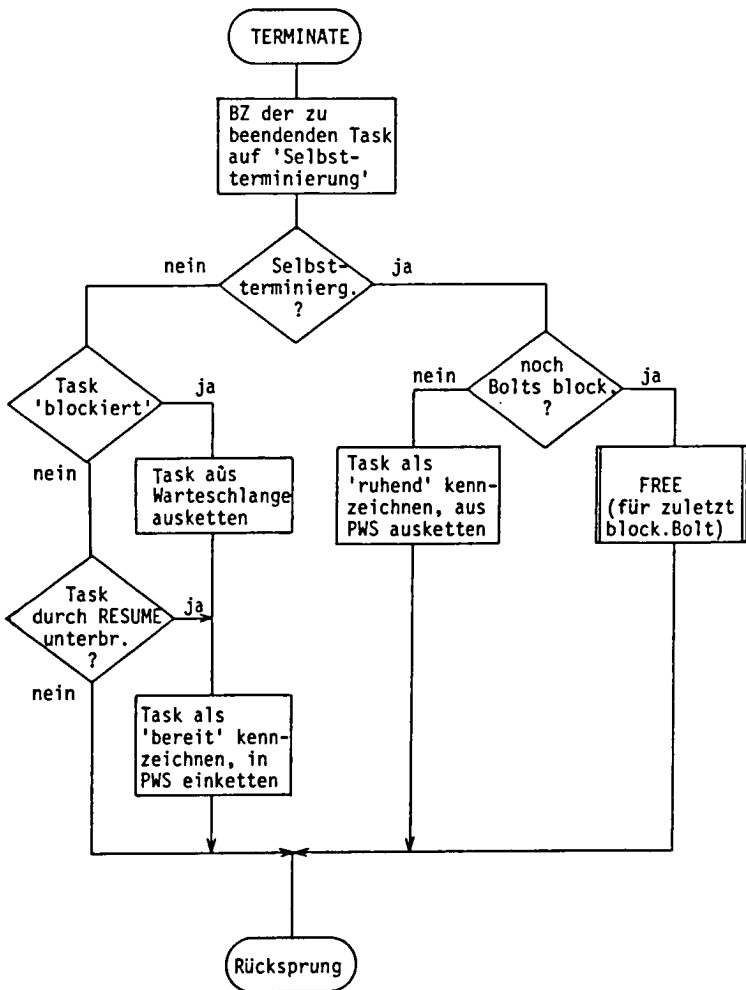


Bild II.21: Taskverwaltung - Modul TERMINATE  
(keine Berücksichtigung 'kritischer Prioritäten')

Ablauffolge ist nur dann in dieser Form erforderlich, wenn das Konzept der 'kritischen Prioritäten' realisiert ist. Die durch den Terminus 'Boltbelegungsgraphen überprüfen' gekennzeichnete Aktion ist relativ umständlich und soll hier nicht im Detail beschrieben werden. Ihre Funktion ist in Kapitel 4.3.5. andeutungsweise erläutert. Sollen kritische Abschnitte keinerlei Einfluß auf Taskprioritäten haben, so stellt sich die Bearbeitung des Aufrufs TERMINATE wie in Bild II.21 dar.

#### II.2.5.3. Der Modul CONTINUE

Der Modul CONTINUE dient im Gegensatz zur Semantik des entsprechenden Aufrufs in PEARL hier nur zur Änderung der Taskpriorität (bzw. der Zeitschranke) zur Laufzeit.

Die durch ein strichliertes Feld gekennzeichnete Aktion in Bild II.22 ist nur dann auszuführen, wenn das Konzept der 'kritischen Prioritäten' zur Anwendung kommt.

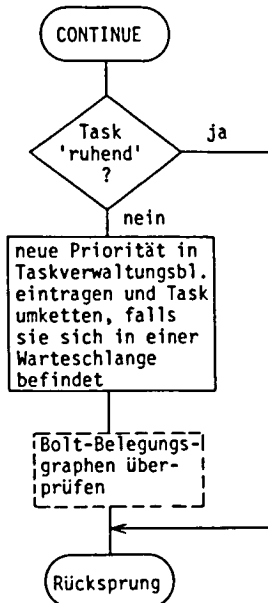


Bild II.22: Taskverwaltung - Modul CONTINUE

#### II.2.5.4. Der Modul RESUME

Der Modul RESUME unterbricht eine Task und sieht ihre, unter den angegebenen Einplanungsbedingungen zu erfolgende Fortsetzung über einen Auftrag an die Einplanungsverwaltung vor. Es besteht zusätzlich die Möglichkeit, die Priorität bzw. Zeitschranke der aufrufenden Task zu verändern.

Die in Bild II.23 durch ein strichliertes Feld gekennzeichnete Befehlsfolge ist nur bei Berücksichtigung von 'kritischen Prioritäten' auszuführen.

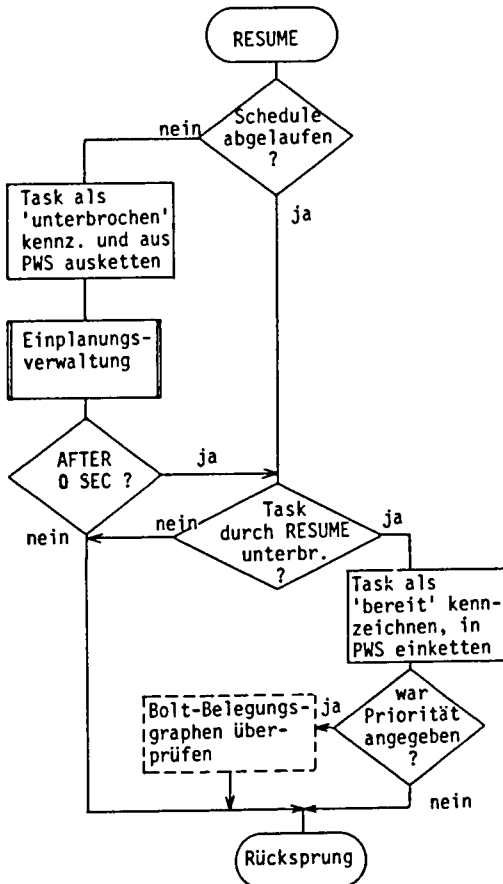


Bild II.23: Taskverwaltung - Modul RESUME

#### II.2.5.5. Der Modul PREVENT

Der Modul PREVENT löscht Einplanungen für

- ACTIVATE, wenn sich der Aufruf auf eine Task bezieht und
- RELEASE, wenn sich der Aufruf auf eine Semaphore bezieht.

Das Löschen eines RESUME-Schedules ist aus folgendem Grunde nicht über PREVENT möglich:

Hat sich eine Task durch RESUME unterbrochen und wird die Fortsetzungseinplanung gelöscht, so kann keine Fortsetzung mehr erfolgen, da keine CONTINUE-Anweisung im Sinne von PEARL realisiert ist. Es ist dann nur mehr eine Beendigung über TERMINATE möglich. Da eine Beendigung einer unterbrochenen Task eine Fortsetzungseinplanung aber ohnedies unwirksam macht, besteht keine zwingende Veranlassung zur expliziten Löscharbeit des Schedules.

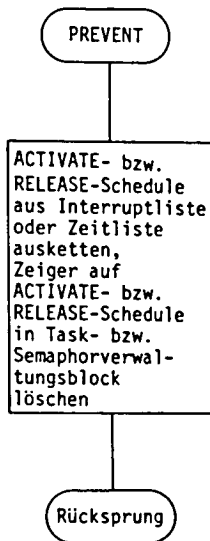


Bild II.24: Taskverwaltung - Modul PREVENT

#### II.2.5.6. Die Moduln REQUEST und RELEASE

Die Moduln REQUEST und RELEASE führen eine Belegung bzw. Freigabe der angegebenen Semaphore aus. Bei RELEASE ist die Angabe von Einplanungsbedingungen möglich.

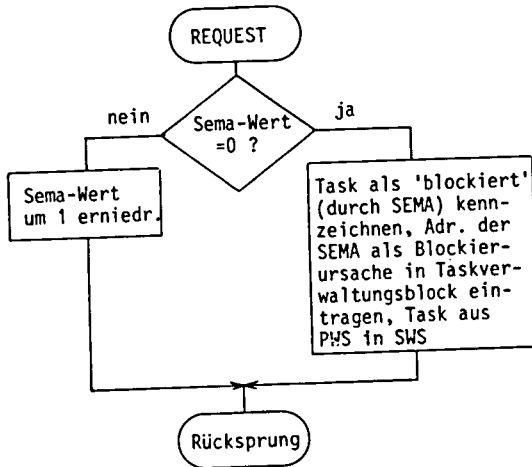


Bild II.25: Taskverwaltung Modul REQUEST

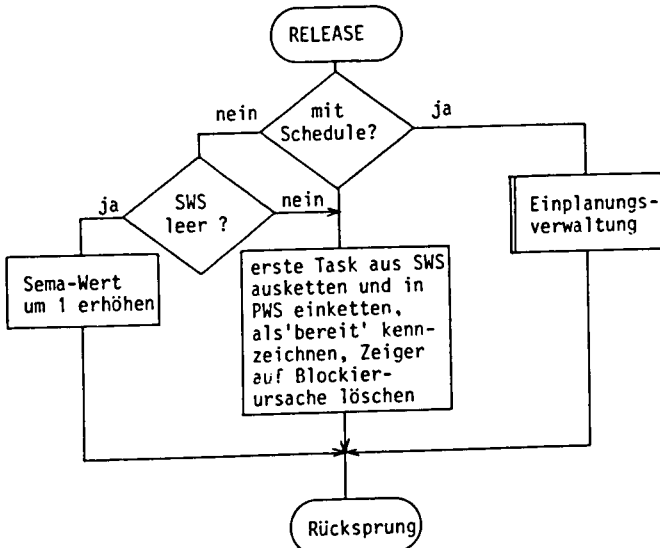


Bild II.26: Taskverwaltung Modul RELEASE



### II.2.5.7. Die Moduln RESERVE und FREE

Die Moduln RESERVE und FREE belegen eine Boltvariable bzw. geben sie wieder frei. Es wird vorausgesetzt (und geprüft), daß die dadurch gebildeten kritischen Abschnitte streng 'geschachtelt' sind, d.h. daß eine Task die zuletzt belegte Boltvariable als erste wieder freigibt. Die in den Flußplänen II.27 und II.28 durch strichlierte Felder gekennzeichnete Befehlsfolgen sind nur im Rahmen des Konzepts der 'kritischen Prioritäten' erforderlich.

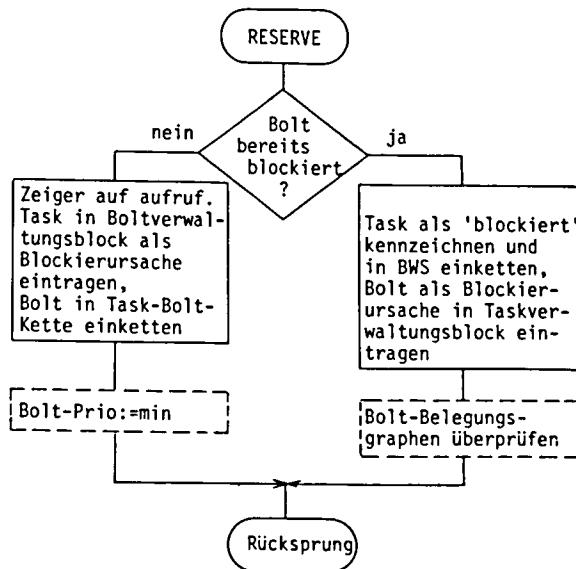


Bild II.27: Taskverwaltung - Modul RESERVE

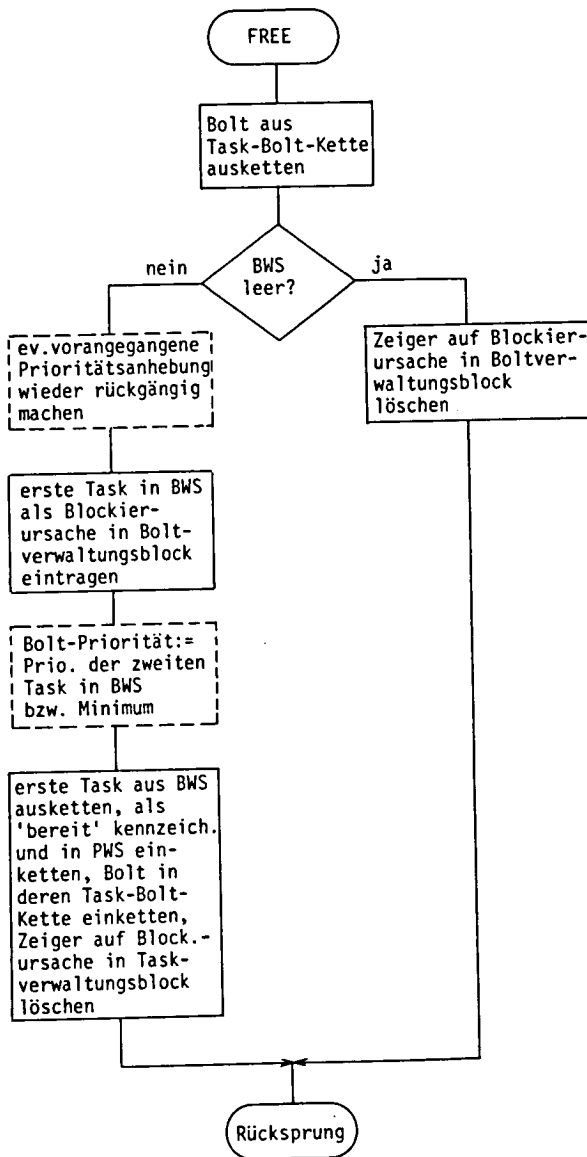


Bild II.28: Taskverwaltung - Modul FREE

### II.2.6. Auftragsübernahme

Die Auftragsübernahme stellt die Schnittstelle zwischen Anwendertasks und Betriebssystem dar.

Eine Anwendertask übergibt der Auftragsübernahme die Adresse eines Auftragsparameterblockes, worüber diese die anzustoßende Verwaltungsroutine identifiziert und aufruft.

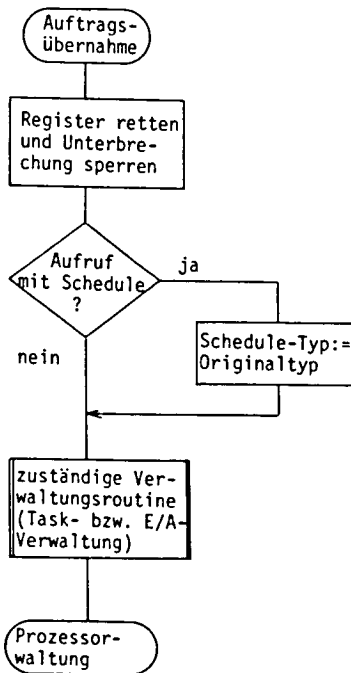


Bild II.29: Flußplan der AUFTRAGSÜBERNAHME

### Lebenslauf

Am 13.10.1945 wurde ich als Sohn des Kaufmanns Gottfried Rössler und seiner Frau Prisca in Linz/Oberösterreich geboren.

In den Jahren 1951 bis 1955 besuchte ich die Volksschule in Aspang/Niederösterreich.

Von 1955 bis 1963 besuchte ich die "Bundesrealschule" in Wr.Neustadt und absolvierte dort die Reifeprüfung im Mai 1963.

Ab Herbst 1963 studierte ich "Allgemeinen Maschinenbau" an der Technischen Hochschule in Wien. Im Jahre 1969 verfaßte ich meine Diplomarbeit mit dem Titel "Optimierung von Kaskadenregelkreisen" und absolvierte im Dezember 1969 die Zweite Staatsprüfung.

Im Januar 1970 wurde ich technischer Angestellter bei der Firma "Nachrichtentechnische Werke AG" (heute SIEMENS AG) in Wien. Bereits im April des gleichen Jahres wurde ich zur Firma SIEMENS AG/Erlangen delegiert und wirkte dort an drei Projekten zur Automatisierung mit Prozeßrechnern mit.

Im Oktober 1972 trat ich meine derzeitige Stellung am III.Physikalischen Institut der Universität Erlangen als wissenschaftlicher Angestellter an. Ich war dort an der Implementation der Programmiersprache PEARL beteiligt und beschäftigte mich dabei insbesondere mit Betriebssystemproblemen sowie mit der Erstellung eines Test- und Bediensystems. Zur Zeit bin ich mit der Erstellung eines Betriebssystemkerns für den Mikroprozessor Z80 im Rahmen des Projekts "PEARL im Nahverkehr" betraut.