RECHENZENTRUM TH MÜNCHEN
ARBEITSGRUPPE BETRIEBSSYSTEME

INTERNSCHRIFT Nr. 32

THEMA: Struktur des Betriebssystems München TR 440 (BSM)

VERFASSER:

Coos

DATUM:

14.11.1969

FORM DER ABFASSUNG

- [x] ENTWURF
- [ ] AUSARBEITUNG
- [ ] ENDFORM

SACHLICHE VERBINDLICHKEIT

- [ ] ALLGEMEINE INFORMATION DISKUSSIONSGRUNDLAGE
- [x] ERARBEITETER VORSCHLAG
- [ ] VERBINDLICHE MITTEILUNG
- [ ] VERALTET

ÄNDERUNGSZUSTAND

BEZUG AUF BISHERIGE INTERNSCHRIFTEN

Vorkenntnisse aus: Hardware-Modelle, 1. - 5. Bericht

Erweiterung von:

Ersatz für:

BEZUG AUF KÜNFTIGE INTERNSCHRIFTEN

Vorkenntnisse zu:

Erweiterung in:

Ersetzt durch:

ANDERWEITIGE LITERATUR

Struktur des Betriebssystems München TR 440 (BSM)

1. Aufgabenstellung

Eine Anlage der Größenordnung des TR 440 hat mehrere Benutzer simultan zu bedienen. Zu diesem Zweck müssen die Betriebsmittel der Anlage, nämlich Prozessoren (Rechnerkerne, E/A-Kanäle, langsame E/A-Geräte) und Speichermedien (Kernspeicher, Massenkernspeicher, Trommel, Platte, Magnetbandgeräte und darauf aufgespannte Magnetbandspulen) entsprechend auf die einzelnen Benutzer verteilt werden. Um Kollisionen (gleichzeitige Anforderung ein und desselben Betriebsmittels durch mehrere Benutzer) zu vermeiden, werden die Betriebsmittel teilweise "maskiert", d.h. für den Benutzer sind gewisse Betriebsmittel in ihrer absoluten Form unzugänglich, es werden ihm nur mit Hilfe der Originalbetriebsmittel definierte neue Betriebsmittel zur Verfügung gestellt. Beispiel: Anstelle von durch Hardware-Adressen adressierbaren Bereichen auf der Platte werden dem Benutzer nur durch symbolische Namen adressierbare Gebiete erlaubt. Der Benutzer erhält also eine "Pseudohardware", welche dynamisch veränderlich ist.

Definition:

Das Programmsystem, welches die dem Benutzer zur Verfügung gestellte Pseudohardware definiert und mit Hilfe der gegebenen Hardware realisiert, heißt Betriebssystem.

Manche Teile der Pseudohardware haben nahezu die gleiche Eigenschaft wie irgendwelche Teile der physikalischen Hardware. Diese Teile der Pseudohardware lassen sich daher durch ein und nur ein physikalisches Betriebsmittel realisieren. Beispiel: Der

./. 2

dem Benutzer zur Verfügung gestellte "virtuelle Kernspeicher" verhält sich im Augenblick des Zugriffs genauso wie der physikalische Kernspeicher. In solchen Fällen besteht die Realisierung der Pseudohardware daher nur in einer Verteilung der physikalischen Hardware auf die einzelnen Benutzer und, eventuell in der Umsetzung von Pseudoadressen in Hardwareadressen. Es ist klar, daß das Betriebssystem nicht mehr physikalische Betriebsmittel verteilen kann, als es selbst zur Verfügung hat. Wird von einem Betriebsmittel mehr angefordert, als im Augenblick verfügbar ist, so kann das Betriebssystem nur mit einer Verschiebung der Antwort auf die Anforderung reagieren. Handelte es sich um Prozessoren, so "wird das System langsamer"; handelte es sich hingegen um eine Anforderung an Speichermedien, so kann das Betriebssystem der Anforderung nur dadurch nachkommen, daß es einem anderen Benutzer Betriebsmittel im gleichen Umfange wegnimmt. Außer für das Betriebsmittel Kernspeicher bedeutet das, daß die Bearbeitung irgend einer Benutzeraufgabe abgebrochen wird und zu einem späteren Zeitpunkt, wenn die Systembelastung geringer ist, wieder aufgenommen wird. Ein solcher Abbruch ist dann besonders kritisch, wenn die abzubrechende Aufgabe nicht separiert werden kann, sondern den Abbruch irgendwelcher anderer Aufgaben nach sich zieht.

Die obige Definition des Begriffs Betriebssystem ist insofern nicht sehr präzise, weil die Frage, was man unter Pseudohardware versteht, davon abhängt, welches Bild der Rechenanlage man dem einzelnen Benutzer vermitteln will. Für einen Benutzer, der beispielsweise nur ALGOL-Programme durchrechnen will, würde der ALGOL-Übersetzer zur Pseudohardware zählen: Der Benutzer hätte eine ALGOL-Maschine. Wir stellen uns hier auf den pragmatischen Standpunkt, daß der Benutzer maschinenintern durch das Telefunken-Programmiersystem vertreten wird. Pseudohardware muß daher genauso beschaffen sein, daß sie den Anschluß des Telefunken-Programmiersystems erlaubt. Zusammengefaßt ergibt das in etwa folgende Anforderungen an die Pseudohardware:

./. 3

2. Anforderungen an die Pseudohardware

2.1 Der Benutzer will einen "Abschnitt" durchrechnen. Die Eingabe dazu erfolgt über den Lochkartenleser, den Lochstreifenleser, Magnetband, usw. (normaler Stapelbetrieb, batchprocessing).

2.2 Der Benutzer will einen Abschnitt durchrechnen. Die Eingabe dazu erfolgt über eine Konsole (Fernschreiber, elektrische Schreibmaschine, Bildsichtgerät). (Remote batch entry.)

2.3 Der Benutzer will eine "Sitzung" an einer Konsole mit Wechselgespräch zwischen Benutzer und Programmen durchführen. (Konversationsbetrieb).

2.4 Der Benutzer will zunächst eine Sitzung durchführen, um mit den erzielten Ergebnissen anschließend einen Abschnitt zu starten (Kopplung von 2.2 und 2.3, 2.2 kommt gewöhnlich nur in dieser Kopplung vor).

2.5 Der Benutzer will in der Zeit zwischen zwei Sitzungen, bzw. Abschnitten, Daten in der Rechenanlage halten, um sie bei späteren Sitzungen, bzw. Abschnitten, wieder zu verwenden. (Datenhaltung vgl. 3).

Erläuterungen

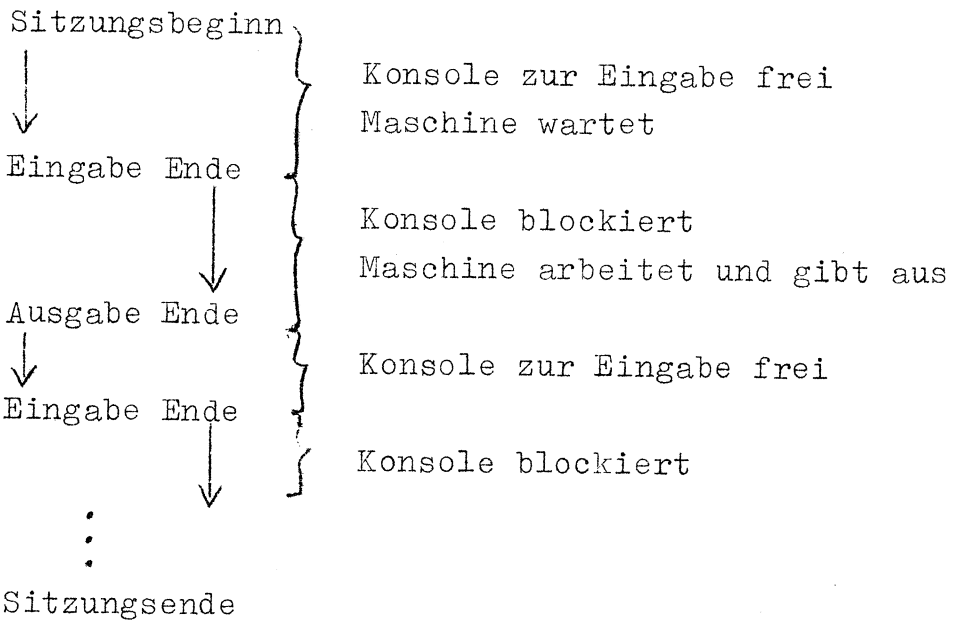
2.6 Unter einem Abschnitt verstehen wir die Ausführung einer linearen Folge von Kommandos, welche im allgemeinen zur Ausführung einer linearen Folge von Programmläufen führt. Die Kommandofolge ist bei Abschnittsbeginn festgelegt und nicht mehr veränderlich. Die Kommandos können unter Bedingung stehen (Beispiel: post mortem Routinen). Ein Abschnitt kann vom Operateur oder vom Benutzer abgebrochen werden. Außerdem kann das Einhängen oder Aushängen von Magnetbändern verlangt sein. Weitere Einflußmöglichkeiten auf den laufenden Abschnitt bestehen im Normalfall nicht. Der Abschnitt muß nicht in dem Augenblick bereits gestartet werden, in dem die zugehörige Eingabe zur Kenntnis des Systems gelangt. Der Start kann vielmehr verzögert werden, wobei Angaben wie Systembelastung, gewünscher Fertigstellungstermin, voraussichtliche Betriebsmittelanfor-

./. 4

derungen usw. berücksichtigt werden.

2.7 Unter einer Sitzung verstehen wir ebenfalls die Ausführung einer linearen Folge von Kommandos. Die Kommandos sind jedoch bei Sitzungsbeginn nicht bekannt, sondern werden während der Sitzung direkt oder indirekt (Aufruf vorprogrammierter Kommandofolgen) an der Konsole eingegeben. Auch die sonstige Eingabe (Programme, Daten) erfolgt on-line während der Sitzung. Da die Konsole normalerweise nur über eine Tastatur als Eingabe verfügt, ist ihre Verwendung für größere Aufgaben nur sinnvoll, wenn Daten aus früheren Abschnitten bzw. Sitzungen wiederverwendet werden können. Bei der Implementierung hat daher die permanente Datenhaltung Vorrang vor der Programmierung der Übertragungsstrecke Konsole ⟷ Benutzerprogramm. Im Gegensatz zum Abschnittsbeginn kann der Beginn einer Sitzung nur kurzfristig (= einige Sekunden) verzögert werden. Ist diese Bedingung wegen Systemüberlastung nicht einzuhalten, so muß der Auftrag "Sitzungsbeginn" vollständig abgelehnt werden.

2.8 Die Sitzung hat die Form eines Wechselgesprächs, bei dem immer nur einer der beiden Beteiligten (Konsolbenutzer, Rechenanlage) sprechberechtigt ist:



Das strenge Wechselgespräch kann durch zwei Ereignisse, die beide Alarmcharakter haben, unterbrochen werden:

2.8.1 Auch wenn die Konsole blockiert ist, kann der Benutzer an der Konsole einen Anrufseingriff (attention interrupt) geben, der eine Alarmmeldung bei dem für den Benutzer rechnenden Programm verursacht. Das Programm wird daraufhin normalerweise baldmöglichst in den Zustand "Erwarte Eingabe" übergehen und die Konsole zur Eingabe freigeben. Der Anrufseingriff ist im allgemeinen ohne Bedeutung, wenn der Zustand "Erwarte Eingabe" bereits vorliegt.

2.8.2 Auch wenn die Konsole zur Eingabe frei ist, kann das Betriebssystem die Konsole blockieren ohne "Eingabe Ende" abzuwaren, beispielsweise um Mitteilungen vom Operateur zu übertragen oder um Fehler im System zu melden, die die weitere Arbeit des Benutzers beeinflussen (oder sogar unmöglich machen) könnten. Auf diese Weise ist auch ein automatischer Abbruch der Sitzung möglich.

2.9 Die Kommandos, welche ein Benutzer während einer Sitzung oder als Teil der Eingabe eines Abschnitts geben kann, zerfallen in 3 Klassen:

2.9.1 Klasse 1: Vermittlerkommandos. Diese steuern die Übertragung der Eingabe (bei Konsolen auch der Ausgabe). Hierher gehören Kommandos, welche den V-Steuereinheiten des TR 4-Systems entsprechen, Umdefinition von Fluchtsymbolen usw. Die wichtigsten Vermittlerkommandos sind jedoch die Kommandos, welche Anfang bzw. Ende der Eingabe eines Abschnitts bzw. den Beginn einer Sitzung bezeichnen. Das Anfangskommando wird außerdem noch als Kommando der Klasse 2, das Endekommando als Kommando der Klasse 3 verstanden; alle anderen Vermittlerkommandos haben nur Bedeutung für die Übertragung.

./. 6

2.9.2 Klasse 2: Systemkommandos. Hier handelt es sich um Kommandos, die der Benutzer geben kann, wenn seine Pseudohardware nicht existiert. Sie werden vom Betriebssystem interpretiert und führen entweder zum Aufbau einer solchen Pseudohardware (Anfangskommando vgl. 2.9.1) oder zur vorsorglichen Bereitstellung von Betriebsmitteln für einen später zu beginnenden Abschnitt oder eine spätere Sitzung (Einhängen einer MB-Spule, Übertragung einer Datenmenge Band→Platte).

2.9.3 Klasse 3: Privatkommandos. Hier handelt es sich um die "normalen" Kommandos, die den Ablauf von Abschnitten oder Sitzungen steuern.

2.9.4 Betrachtet man den Operateur als einen Benutzer, dessen Pseudohardware sich allerdings wesentlich von der anderer Benutzer unterscheidet, so sind die Operateurkommandos überwiegend zur Klasse 3 zu zählen. Ausnahmen: Eingaben von der Operateurkonsole im Wartungsbetrieb, während des Systemaufbaus, sowie zum Warmstart des Systems, wenn die Pseudohardware des Operateurs funktionsunfähig ist.

2.10 Der interne Unterschied zwischen Stapel- und Konversationsbetrieb

Die Pseudohardware eines Benutzers hat eine eigene "Pseudozeit", die, in realer Zeit gemessen, nur während der Zeitintervalle fortschreitet, in denen ein physikalischer Rechnerkern der Pseudohardware dieses Benutzers zugewiesen ist. Während die Zuteilung solcher Zeitintervalle bei Abschnitten beliebig vorgenommen werden kann, solange dem keine Terminangaben entgegenstehen, ist bei Sitzungen zu berücksichtigen, daß an den Konsolen Menschen sitzen, denen eine "vernünftige" Antwortzeit garantiert werden muß, wenn ein sinnvolles Wechselgespräch zustandekommen soll. Im wesentlichen heißt das, daß für keinen Abschnitt Rechenzeit zur Verfügung gestellt werden darf, so lange Sitzungen auf Rechenzeit warten. (Das

./. 7

ist die Definition von "vernünftige Antwortzeit"). Abgesehen von der verschiedenartigen Ein/Ausgabe ist diese unterschiedliche Behandlung von Abschnitten und Sitzungen bei der Zuteilung von Rechenzeit (und damit auch des dabei benötigten Kernspeichers) das Hauptmerkmal, welches Stapel- und Konversationsbetrieb unterscheidet.

Sollte jedoch die Betriebsmittelforderung eines Benutzers im Konversationsbetrieb zu groß werden, so muß man diesen Benutzer bei der Zuteilung von Betriebsmitteln ebenfalls mit geringerer Priorität berücksichtigen, um andere Benutzer nicht über Gebühr zu benachteiligen. Es gibt also einen gleitenden Übergang zwischen Konversationsbetrieb und Stapelbetrieb. Überlegungen über die bevorzugte Behandlung von Benutzern im Konversationsbetrieb gehen aus von einer Rechenzeit in der Größenordnung einige hundert msec. zwischen Ende der Konsoleingabe und Beginn der nächsten Konsolausgabe.

Die Diskussion zeigt außerdem den Unterschied zwischen Echtzeitbetrieb und Konversationsbetrieb: Beim Echtzeitbetrieb ist eine Antwortzeit in der Größenordnung einige Millisekunden bis Sekunden vorgeschrieben. Wird sie nicht eingehalten, so ist die Antwort wertlos. Beim Konversationsbetrieb wird nur eine "vernünftige Antwortzeit" verlangt. Bei Verlangsamung des Systems wegen Überlastung kann diese Antwortzeit größer werden, ohne daß die Antwort in meßbarer Weise an Wert verliert.

./. 8

3. Betriebsmittel aus der Sicht des Benutzers

Ein Benutzer unterscheidet im wesentlichen 3 Betriebsmittel: Rechnerkern (Rechenzeit), Arbeitsspeicher (Kernspeicher), Gebiete. Rechenzeit erhält er automatisch unter Berücksichtigung der in 2.10 diskutierten Gesichtspunkte. Arbeitsspeicherbereiche (Segmente) und Gebiete müssen explizit geschaffen bzw. beseitigt werden. Bei der Schaffung sind eine Reihe von Spezifikationen mitzuliefern.

Bei Segmenten sind das etwa: Angaben über Schreibschutz, Angaben, ob das Segment von anderen Benutzern mitbenutzt werden soll und wenn ja von welchen, Angaben darüber, ob das Segment vorbesetzt werden soll oder nicht, eventuell eine Lagebeschreibung, welche implizit die voraussichtliche Häufigkeit des Zugriffs angibt und das System befähigt, zu entscheiden, ob das Segment im Massenkernspeicher untergebracht werden darf oder nicht.

Gebiete sind Datenmengen, auf die nur satzweise, nicht aber im Einzelwortzugriff zugegriffen werden kann. Bei Schaffung eines Gebiets sind folgende Angaben notwendig:

Unterscheidung permanentes Gebiet / temporäres Gebiet (u.a. scratch file).

Paßwortspezifikation und Angabe der sonstigen Benutzer, welche das Gebiet mitbenutzen dürfen.

Angaben über die Zugriffsberechtigung von Mitbenutzern (Lese- oder Schreiberlaubnis, Schreiberlaubnis im Mehrfachzugriff).

Qualitätsangabe (interne Gebiete (Platte, Trommel-Gebiete), Eingabegebiet von LKL, LSL, Ausgabegebiet für DR, LKS, LSS, Magnetbandgebiete, Konsoleingabe- und -ausgabegebiete).

Gebiete bilden die Grundlage der Dateiorganisation des Telefunken-Programmiersystems.

./. 9

Zwischen Erschaffung und Beseitigung kann ein Gebiet mehrfach angemeldet (eröffnet) und danach abgemeldet (geschlossen) werden. Bei der Anmeldung wird jeweils die Zugriffsberechtigung und, falls vorhanden, das Paßwort überprüft.

Bei Anmeldung, Abmeldung und in der Dateiorganisation ist Rücksicht auf die Qualitätsangabe zu nehmen:

Bei Anmeldung eines Eingabegebiets wird das Gebiet vom Betriebssystem angefordert, wenn es noch nicht vorhanden ist.

Bei Abmeldung eines Ausgabegebiets wird dieses an das Betriebssystem zum Zweck des Ausdruckens bzw. Ausstanzens weitergegeben.

Magnetbandgebiete sind nur zur Benutzung als sequentielle Dateien zugelassen.

Die Anmeldung eines Konsoleingabe- bzw. -ausgabegebiets erfordert die Überprüfung der Existenz einer Übertragungsstrecke Konsole - Benutzerprogramm. Lesen bzw. Schreiben eines Satzes auf einem Konsoleingabe- bzw. -ausgabegebiet bewirken unmittelbar den Transport von bzw. zur Konsole. Als Gebiete werden sie nur deshalb bezeichnet, weil das aktuelle Benutzerprogramm keinen Unterschied zwischen gewöhnlichen Eingabegebieten und Konsoleingabegebieten kennen muß. Das Anmelden und Abmelden von Magnetbandgebieten bewirkt eine Aufforderung an den Operateur, die entsprechende Magnetbandspule auf- bzw. abzuspannen.

Bei Platzmangel auf der Platte kann das Betriebssystem Gebiete von der Platte auf Magnetband auslagern. Die Anmeldung eines ausgelagerten Gebiets führt zwar zur Aufforderung an den Operateur, das Gebiet wieder auf die Platte zu bringen, dem Benutzer wird jedoch eine Fehlermeldung gegeben, die das Gebiet als augenblicklich unzugänglich charakterisiert. Ausgelagerte Gebiete und Magnetbandgebiete verhalten sich also verschieden.

./. 10

Die Unterscheidung permanentes Gebiet / temporäres Gebiet ist nur bei internen und Magnetbandgebieten von Bedeutung. Temporäre Gebiete werden bei Sitzungs- bzw. Abschnittsende automatisch beseitigt.

Der Benutzer, welcher ein Gebiet eröffnet, heißt Eigentümer des Gebiets. Ihm werden die Kosten für die Unterbringung des Gebiets angelastet. Meldet ein Benutzer ein Gebiet an, bei dem er nicht Eigentümer ist, so heißt das Gebiet vom Augenblick der Anmeldung an bis zur Abmeldung ein Fremdgebiet dieses Benutzers.

./. 11

4. Bearbeiter

4.1 Normal-Bearbeiter

Die vom BS für einen bestimmten Benutzer zu erledigenden Aufgaben lassen sich grob unterteilen in lokale und globale Aufgaben. Lokale Aufgaben sind solche, die völlig separiert von den Aufgaben anderer Benutzer bearbeitet werden können. Globale Aufgaben sind solche, bei denen Interferenzen mit Aufgaben anderer Benutzer erwartet werden müssen. Letzteres ist bei allen Arten von Betriebsmittelanforderungen bzw. Freigaben, sowie bei Transporten von und zur Trommel, Platte, TR 86 usw. der Fall. Entsprechend ist das BS in einen lokalen Teil, den benutzerspezifischen Abwickler und einen globalen Teil, das eigentliche Betriebssystem, unterteilt.

Im weiteren verstehen wir unter "dem Betriebssystem" grundsätzlich nur noch das eigentliche Betriebssystem.

Obwohl theoretisch jeder Benutzer seinen eigenen Abwickler mitbringen kann, gehen wir bei quantitativen Abschätzungen davon aus, daß es nur sehr wenige verschiedene Abwickler-Typen gibt.

Ein Abwickler und alle (unter seiner Kontrolle) ablaufenden (Benutzer-)Programme erscheinen aus der Sicht des eigentlichen BS als eine organisatorische Einheit. Jeder solchen Einheit wird ein Hardware-Leitblock zugeteilt. Der Rechnerkern arbeitet auf dieser Einheit grundsätzlich im Abwickler- oder Normalmodus.

Ohne Rücksicht auf den speziellen Abwickler und die augenblicklich laufenden Programme bezeichnen wir diese Einheit als einen Normal-Bearbeiter (N-Bearbeiter). Ähnlch wie wir unter einer "Prozedur" in höheren Programmiersprachen einerseits eine Organisationsform für einen Programmmodul verstehen, andererseits aber einen konkreten Programmmodul, der diese Organisationsform aufweist, bezeichnet der Begriff "Normal-Bearbeiter" einerseits

./. 12

eine Organisationsform, die durch die Festlegungen "hat eigenen Leitblock", "läuft im Abwickler- oder Normalmodus", sowie durch ein bestimmtes Verzeichnis von Schnittstellen zum eigentlichen BS charakterisiert ist, andererseits verstehen wir unter einem Normal-Bearbeiter eine Kollektion von Programmen und Daten, welche diese Organisationformen aufweisen. Die erste Interpretation ist diejenige, welche aus der Sicht des Benutzers zu bevorzugen ist - der N-Bearbeiter definiert im wesentlichen die Pseudohardware des Benutzers -. Die zweite Interpretation ist die Auffassung des System-Konstrukteurs.

Ein Benutzer bedarf eines Bearbeiters lediglich während der Zeit, in der ein Abschnitt oder eine Sitzung abläuft. In dieser Zeit wird der Benutzer maschinenintern durch den Bearbeiter repräsentiert. In der Zeit zwischen zwei Abschnitten oder Sitzungen bedarf es keines Bearbeiters. Es genügt ein Verzeichnis der permanenten Gebiete, das Guthabenkonto, aus dem Betriebsmittel zu bezahlen sind, das Benutzer-Paßwort, das zur Eröffnung eines neuen Abschnitts oder einer neuen Sitzung zu nennen ist, und ähnliche Informationen. Wir fassen diese Informationen unter dem Begriff "permanente Benutzerinformation" zusammen. Wir legen fest:

N-Bearbeiter des bisher besprochenen Typs haben eine Lebensdauer, welche der Zeit eines Abschnitts oder einer Sitzung entspricht. Das Systemkommando "Abschnittsbeginn" bzw. "Sitzungsbeginn" hat daher intern die Form
KONSTRUIERE BEARBEITER (PROGRAMMNAME)
Der Programmname ist der Name des Abwicklers, der in den Bearbeiter einzubetten ist und der durch sein Urstartprogramm den weiteren Aufbau des Bearbeiters veranlaßt. Der Programmname wird der permanenten Benutzerinformation entnommen.
Das Kommando Abschnittsende bzw. Sitzungsende veranlaßt den Abwickler, einen definierten Grundzustand herzustellen und anschließend mit dem Auftrag
LOESCHE BEARBEITER
an das BS seine Arbeit einzustellen.

./. 13

Diese Festlegungen haben u.a. folgende Konsequenzen:

(a) Ein Benutzer kann nicht zur gleichen Zeit einen Abschnitt und eine Sitzung durchführen, ohne aufwendige Koordinierungsmaßnahmen bei der Benutzung seiner eigenen permanenten Gebiete zu ergreifen.

(b) Die Herstellung des Grundzustands, in dem der Bearbeiter gelöscht werden kann, macht zwar normalerweise keine Schwierigkeiten. Solche entstehen jedoch, wenn die Bearbeitung infolge unvorgesehener Ereignisse (Zeitüberschreitung eines Operatorlaufs, Systemfehler u.ä.) gestoppt, und der augenblickliche Zustand des Bearbeiters sichergestellt werden muß, um die Bearbeitung in einem späteren Abschnitt (Sitzung) an der gleichen Stelle wieder aufnehmen zu können.

4.2 Bearbeiter und sequentielle Prozesse

Da aus Hardware-Gründen auf einem Leitblock zu jedem Zeitpunkt höchstens ein Rechnerkern arbeiten kann, erscheint ein N-Bearbeiter zumindest für die Rechnerkernvergabe stets als ein sequentieller Prozeß. Für andere Teile des eigentlichen Betriebssystems könnte jedoch der N-Bearbeiter als eine Kollektion mehrerer sequentieller Prozesse erscheinen, wenn man zuläßt, daß während der Zeit, in der dem Bearbeiter kein Rechnerkern zugeteilt ist, der Zustand, in dem der N-Bearbeiter das nächste Mal den Rechnerkern übernehmen soll, (nahezu) beliebig umdefiniert wird. BS1 verfolgt diese Richtung. Wir nehmen hier den gegenteiligen Standpunkt ein, daß, zumihdest im Prinzip, der N-Bearbeiter für das gesamte eigentliche Betriebssystem als ein sequentieller Prozeß erscheinen sollte. ("Im Prinzip" beduetet wie üblich: Ausnahmen (nämlich Alarme, vgl. 4.2.1) bestätigen die Regel.) Die Gründe sind folgende:

(a) In einem Mehrprozessorsystem muß dem Benutzer auf jeden Fall die Möglichkeit geboten werden, mehrere sequentielle Prozesse echt parallel durchzuführen. Das bedeutet, daß ihm mehrere Bearbeiter zugeordnet werden können. Daraufhin erscheint es überflüssig, zusätzlich mehrere sequentielle Prozesse in einem Bearbeiter zuzulassen.

(b) Entwurf und Programmierung sowohl von Abwicklern als auch des BS werden logisch übersichtlicher und durchsichtiger. Bei mehreren sequentiellen Prozessen in einem Bearbeiter besteht die Gefahr, daß schwerwiegende logische Fehler im Zusammenspiel Bearbeiter-BS nur durch unvertretbar hohen Testaufwand beseitigt werden können, sowie daß die Konventionen für die Konstruktion von Abwicklern so kompliziert werden, daß letztlich doch niemand von der Möglichkeit mehrerer Prozesse Gebrauch macht.

Man beachte, daß diese Konvention natürlich nicht Multiprogramming, also eine Aufspaltung in mehrere quasisimultan ablaufenden Prozesse, innerhalb des Bearbeiters verbietet, solange die Schnittstelle Bearbeiter-BS davon unberührt bleibt.

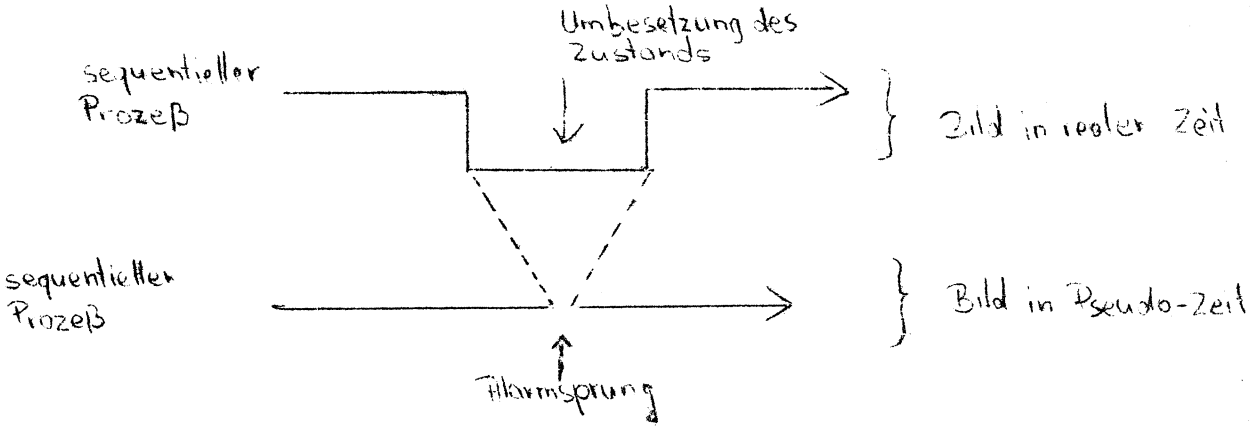
4.2.1 Alarme

Der Begriff "sequentieller Prozeß" kennzeichnet einen dynamischen Ablauf, der aus einer linearen Folge von Elementaroperation besteht, und bei dem aus dem "Zustand" (den internen Variablen, Registerinhalten und dem Befehlszählerstand) die nächste Elementaroperation eindeutig vorhergesagt werden kann. Alarme durchbrechen diese determinierte Operationsfolge. Zumindest bei äußerlicher Betrachtungsweise erscheint ein Alarm als unvorhergesehene Abweichung von der bisherigen Operationsfolge. Betrachten wir jedoch etwa einen arithmetischen Alarm mit dem Detaillierungsgrad der Mikroprogrammierung, so erweist sich der Sprung auf eine Alarmadresse als normaler Bestandteil des sequentiellen Ablaufs: Das Alarmbit wurde durch irgendeine der vorangehenden Elementaroperationen gesetzt und der den Alarm auslösende Befehl wurde vom Leitwerk interpretiert als

if arith. Alarmbit then goto Alarmadresse else
führe Befehl aus fi

Der Sprung ist in Wirklichkeit verbunden mit einer Abspeicherung des augenblicklichen Leitwerk-Zustands, der Erzeugung eines Alarmwortes, das angibt, welches der verschied Alarmbits gesetzt war, sowie mit dem Setzen einer Alarmsperre. Der Sprung hat also einen Nebeneffekt. Obwohl diese Interpretation zunächst nur auf tatsächliche Hardware-Alarme anwendbar ist, können wir den gleichen Effekt auch durch das Betriebs-

system erreichen: Während der sequentielle Prozeß (in realer Zeit gerechnet) unterbrochen ist, kann man seine Zustandsbits so umbesetzen, daß bei späterer Fortsetzung die Unterbrechung als die Ausführung eines Alarmsprungs erscheint. Dieser ist nicht zu unterscheiden von einem echten Alarmsprung, der aufgrund von Hardware-Bedingungen eintritt. Aus der Sicht des Benutzers erscheint beides als Alarm in seiner Pseudo-Hardware.

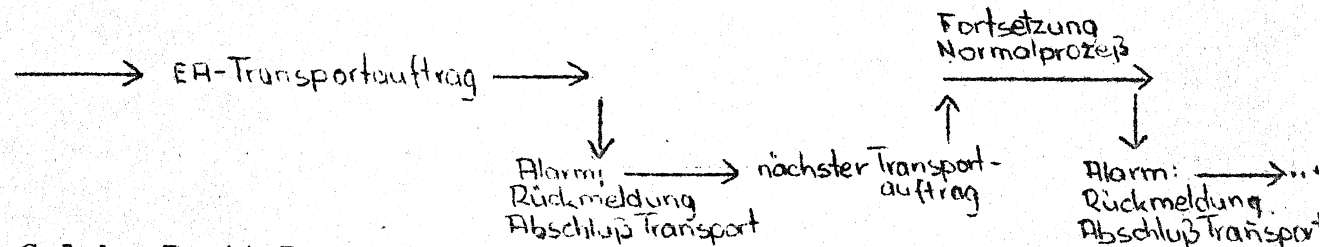


Diese Überlegungen zeigen, daß die folgende Festlegung möglich (wenn auch etwas mühsam) ist: Ein durch Hardware oder das Betriebssystem ausgelöster Alarm bei einem Bearbeiter wird verstanden als ein Sprung mit Nebeneffekt, der im Rahmen des auf dem Bearbeiter ablaufenden sequentiellen Prozesses bleibt. Das Betriebssystem sollte programmierte Alarme nur verursachen, wenn diese echten Alarmcharakter für den betroffenen sequentiellen Prozeß haben, oder, wenn dies aus anderen Gründen unumgänglich Derzeit bekannt sind folgende Arten programmierter Alarme:
Anruf, d.h. ein vom BS dem N-Bearbeiter zugestellter Anrufseingriff.
Zeitmeldungen.
Systemfehlermeldungen, d.h. Meldungen über leichtere Systemzusammenbrüche, die zwar nicht den Abbruch des sequentiellen Prozesses erzwingen, ihm aber trotzdem mitgeteilt werden müssen.

Nicht als Alarme zu interpretieren sind hingegen Meldungen über fehlerhafte Ausführung einer Systemanfrage oder Meldungen über den Abschluß eines E/A-Transports. In beiden Fällen handelt es sich um die Beendigung einer Pseudo-Instruktion der Pseudohardware des Benutzers, die aus der Sicht des Benutzers genausowenig einen Alarm rechtfertigt wie die Rückkehr aus einem Unterprog-

./. 16

Die obige Festlegung in Zusammenhang mit der Forderung, daß das Betriebssystem Rückmeldungen zu Systemanfragen nicht durch Alarme realisiert, schließt die Möglichkeit aus, daß jemand "unter der Hand" über ein Alarmprogramm einen zweiten sequentiellen Prozeß ablaufen läßt, etwa in der Form



Solche Zweit-Prozesse sollen nur über die in 4.4 zu besprechenden Neben-Bearbeiter möglich sein.

4.3 S-Bearbeiter

Das eigentliche Betriebssystem enthält eine Reihe von sequentiellen Prozessen, die sich organisatorisch ähnlich wie die auf N-Bearbeitern ablaufenden Prozesse verhalten. Um Einheitlichkeit zu erzielen, versucht man daher für alle "Systemprozesse", für die das möglich ist, eine Organisationsform zu finden, welche sich möglichst eng an die Organisationsform N-Bearbeiter anlehnt. Wir sagen dann "der Systemprozeß läuft auf einem System-Bearbeiter (S-Bearbeiter)". Im Gegensatz zu N-Bearbeitern, die eine einheitliche Schnittstelle gegenüber dem eigentlichen Betriebssystem haben, kann von einer solchen einheitlichen Schnittstelle bei S-Bearbeitern nicht mehr die Rede sein, andernfalls könnten die S-Bearbeiter ihre Aufgaben nicht erfüllen. Die allen S-Bearbeitern gemeinsamen Eigenschaften sind:

Sie sind selbständige Konkurrenten bei der Rechnerkern-Vergabe.

Sie besitzen einen Leitblock. Für den Fall, daß der S-Bearbeiter im Systemmodus läuft, enthält der Leitblock keine Kacheltabelle und ist eine reine Software-Angelegenheit (Pseudo-Leitblock)

Die Kommunikation zwischen S-Bearbeitern gehorcht einheitlichen Regeln.

Soweit S-Bearbeiter dynamisch Speicher anfordern dürfen, verhalten sie sich dabei genau wie N-Bearbeiter; Ausnahme: S-Bearbeiter dürfen nicht nur 1024-Wort-weise, sondern auch 128-Wort-weise Speicher anmelden.

S-Bearbeiter sind permanent. Sie werden nicht nach Beendigung eines Auftrags gelöscht, wie das mutatis mutandis bei N-Bearbeitern der Fall ist.

Ein "Bearbeiter" ist entweder ein S-Bearbeiter oder ein N-Bearbeiter. Der auf einem Bearbeiter ablaufende sequentielle Prozeß heißt eine "Bearbeitung" oder kurz ein "Prozeß". Während ein N-Bearbeiter nach Abschluß der Bearbeitung gelöscht wird, laufen auf einem S-Bearbeiter viele Bearbeitungen hintereinander ab, wenn wir die Erledigung eines einzelnen Auftrags als eine Bearbeitung ansehen.

4.4 Neben-Bearbeiter

Neben-Bearbeiter sind spezielle N-Bearbeiter. Sie eröffnen die Möglichkeit, die in einer Sitzung oder einem Abschnitt anfallenden Aufgaben auf mehrere, auch aus der Sicht des eigentlichen BS unabhängige, sequentielle Prozesse zu verteilen. Während gewöhnliche N-Bearbeiter, wie in 4.1 beschrieben, durch Systemkommandos (also im Auftrag eines S-Bearbeiters) geschaffen werden, kommt der Auftrag zur Konstruktion eines Neben-Bearbeiters stets von einem N-Bearbeiter, ansonsten hat der Auftrag die gleiche Form wie in 4.1 beschrieben. Entsprechend der speziellen Funktion eines Neben-Bearbeiters, sind diesem unmittelbar nach Konstruktion weitere Programme und Daten-Gebiete zu übergeben, die seinen Auftrag näher spezifizieren; außerdem ist normalerweise ein "Ereignis-Kanal" (siehe den späteren Abschnitt über den Zusteller) zu definieren, über den der "Vater-Bearbeiter" und der "Tochter-Bearbeiter" miteinander kommunizieren können.

./. 18

Da ein Neben-Bearbeiter keinen Benutzer repräsentiert, sind seine Möglichkeiten bezüglich Gebiets-Operationen eingeschränkt: Er kann sich z ar nach Belieben temporäre und Fremd-Gebiete (letztere z.B. von seinem Vater-Bearbeiter) beschaffen, jedoch ist ihm die Eröffnung permanenter Gebiete untersagt. Die durch einen Neben-Bearbeiter verursachten Kosten gehen zu Lasten des durch den Vater-Bearbeiter repräsentierten Benutzers.

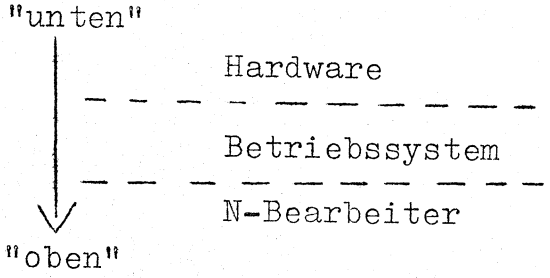
Natürlich sind auch Neben-Bearbeiter in der Lage, weitere Neben-Bearbeiter zu schaffen, so daß man einen ganzen Baum von Bearbeitern erhält. Diese Möglichkeit dürfte jedoch praktisch ohne Interesse sein. Bei praktischen Überlegungen gehen wir von der Annahme aus, daß der in einem Neben-Bearbeiter enthaltene Abwickler gegenüber einem gewöhnlichen N-Bearbeiter stark vereinfacht ist, und daß die Kernspeicherbereiche des Neben-Bearbeiters überwiegend gemeinsame Bereiche von Vater- und Tochter-Bearbeiter sind. Dies ist etwa der Fall, wenn man PL/I-Multi-Tasking oder die "collateral-clauses" von ALGOL 68 mittels mehrerer Bearbeiter realisiert. Es ist darauf zu achten, daß die Benutzung von Neben-Bearbeitern in erster Linie einfach zu sein hat, unter Verzicht auf überflüssige Allgemeinheit; andernfalls sind Neben-Bearbeiter praktisch nicht existent (vgl. das Schicksal der Nebenstufen im TR4-System).

./. 19

5. Allgemeine Strukturprinzipien

5.0 Freie Unterprogramme und ihre Verwendung.

Aus dem bisherigen ergibt sich folgende Gliederung des Gesamtsystems:



Organisatorisch besteht dabei das Betriebssystem aus Einheiten folgender Typen:

5.0.1 Programme, welche die Grundlage für die Organisationsform "S-Bearbeiter" liefern (z.B. die Rechnerkernvergabe, allgemeiner: alle Programme, welche auf die Anzahl der Hardware-Rechnerkerne im System explizit Bezug nehmen).

5.0.2 S-Bearbeiter.

5.0.3 Unterprogramme, welche Dienstleistungen für irgendwelche Bearbeiter erbringen, organisatorisch jedoch nicht zu diesen Bearbeitern gehören. Die Feststellung "gehört organisatorisch nicht zum aufrufenden Bearbeiter" ist per definitionem genau dann zutreffend, wenn das Unterprogramm durch einen SSR-Befehl im Abwickler-, Spezial- oder System-Modus aufgerufen wird. Wir unterscheiden dabei folgende Unterfälle:

5.0.3.1 Das Unterprogramm erbringt die Dienstleistung unmittelbar.

5.0.3.2 Das Unterprogramm beauftragt einen S-Bearbeiter, die gewünschte Dienstleistung zu erbringen. Das Unterprogramm selbst wartet auf ein Ereignis (V(Semaphor)), das durch den gestarteten sequentiellen Prozeß ausgelöst wird und kehrt dann zurück. Aus der Sicht des aufrufenden Bearbeiters sind die Fälle 5.0.3.1 und 5.0.3.2 identisch.

./. 20

5.0.3.3 Das Unterprogramm beauftragt einen S-Bearbeiter, die gewünschte Dienstleistung zu erbringen und kehrt dann zurück.

5.0.3.4 Das Unterprogramm erwartet ein Ereignis, ausgelöst durch den sequentiellen Prozeß, welcher durch einen Aufruf des Typs 5.0.3.3 beauftragt wurde, und kehrt dann zurück (Zielabfrage). 5.0.3.2 kann verstanden werden als Hintereinanderausführung von Unterprogrammen der Form 5.0.3.3 und 5.0.3.4.

Anmerkungen zu 5.0.3

5.0.3.5 Die Unterprogrammform 5.0.3.2 ist meist nicht in reiner Form gegeben. Gewöhnlich hat das Unterprogramm zu entscheiden, ob 5.0.3.1 oder 5.0.3.2 vorliegt.

5.0.3.6 Die Zerlegung, etwa eines Transportauftrages in 2 Unterprogrammaufrufe der Form 5.0.3.3 und 5.0.3.4 ist einerseits eine der Grundlagen jeder Aufteilung von Aufgaben auf mehrere Prozesse. Andererseits ist sie sehr gefährlich: Folgt die zugehörige Zielabfrage nicht, oder nicht rechtzeitig, auf den durch einen Aufruf der Form 5.0.3.3 gegebenen Auftrag, so könnten erhebliche Betriebsmittel für längere Zeit gebunden werden, ohne daß sie aktuell benutzt werden. Beispiel: Jemand verlangt das Aufspannen einer MB-Spule und vergißt anschließend, etwa wegen Programmierfehlers, die Spule zu benutzen oder wenigstens das MB-Gerät wieder freizugeben. In solchen Fällen muß das System das Recht haben, nach einer bestimmten Zeit das Betriebsmittel ohne Notiz zurückzufordern oder wenigstens dem Bearbeiter einen Alarm zuzustellen. Wir unterscheiden in dieser Hinsicht folgende Auftragstypen, die mit 5.0.3.3, 4 behandelt werden:

5.0.3.7 Aufträge, bei denen dem Rest des Systems trotz fehlender Zielabfrage keine weitere Behandlungsmöglichkeit offensteht. Dies ist der gefährlichste Typ, welcher normalerweise den Benutzern nicht zugänglich gemacht werden darf.

./. 21

5.0.3.8 Aufträge, bei denen eine fehlende Zielabfrage zur Freigabe ohne weitere Notiz führt. Eine zu spät erfolgende Zielabfrage kann eventuell noch positiv beantwortet werden, meist führt sie jedoch zu einer Fehlermeldung. (Beispiel: Auftrag, ein Gebiet am Drucker auszugeben.)

5.0.3.9 Aufträge, bei denen die Zielabfrage innerhalb einer bestimmten Zeit (Pseudo-Zeit des Auftraggebers) nach dem Eintreffen des abzufragenden Signals kommen muß, widrigenfalls beim Auftraggeber ein Alarm verursacht wird.

5.0.3.10 Um die Zugänglichkeit der Unterprogramme individuell steuern zu können, so daß zum Beispiel Systemprogrammierer größere Rechte erhalten können als sonstige Benutzer, wird vorgeschlagen, die Unterprogramme des BS (d.h. die SSR-Befehle) in n Gruppen einzuteilen, und jedem Bearbeiter einen "SSR-Schlüssel" zuzuordnen, welcher die Gruppen spezifiziert, aus denen der Bearbeiter Unterprogramme auswählen darf.

5.0.3.11 Die Regelung von 5.0.3.10 gilt auch für diejenigen Unterprogrammaufrufe, welche Aufträge an die in 5.0.1 genannten Programme abgeben, bzw. deren Ende erwarten. Überhaupt zeigt sich, daß in 5.0.3.2 - 5.0.3.4 die Annahme, daß ein S-Bearbeiter beteiligt ist, unwesentlich ist: Es könnte auch ein sequentieller Prozeß auf einer Ebene unterhalb der S-Bearbeiter (z.B. ein E/A-Vorgang) oder im anderen Extremfall ein N-Bearbeiter beauftragt werden.

5.0.3.12 Die vorstehende Diskussion zeigt, daß Unterprogramme der in 5.0.3.2 - 5.0.3.4 angegebenen Typen zwar nicht dem aufrufenden Bearbeiter zugeordnet werden können; andererseits stehen sie mit dem beauftragten S-Bearbeiter in so engem, funktionellen Zusammenhang, daß man sie als Start- bzw. Ziel-Unterprogramme für den jeweiligen S-Bearbeiter bezeichnen kann. Wir gelangen in diesem Fall somit zu folgender Vorstellung:

Ein Bearbeiter verlangt eine Dienstleistung des BS durch Aufruf eines Start-Unterprogramms; das Start-Unterprogramm beauftragt einen S-Bearbeiter und kehrt zurück; der Bearbeiter verlangt die Beendigung der Dienstleistung durch Aufruf eines Ziel-Unterprogramms; das Ziel-Unterprogramm fragt das erwartete Ereignis ab und wartet eventuell auf dessen Eintreten, danach kehrt es zurück.

"Das Ziel-Unterprogramm wartet" heißt in Wirklichkeit: "Der anfragende Bearbeiter wartet, während der Ausführung des Ziel-Unterprogramms". In seiner eigenen Pseudo-Zeit wird dem Bearbeiter dieses Warten jedoch nicht bekannt. Insbesondere könnte das Ziel-Unterprogramm mehrmals von verschiedenen Bearbeitern aufgerufen werden, während ein Bearbeiter immer noch auf die Beendigung des Ziel-Unterprogramms wartet.

5.1 Prozeß-Kommunikation

Aus dem bisherigen, speziell aus 4.2.1 und 5.0 ergeben sich folgende Möglichkeiten, Information zwischen sequentiellen Prozessen zu übertragen.

5.1.1 Alarme (dieses Hilfsmittel soll nur in Ausnahmefällen benutzt werden).

5.1.2 Aufruf von Start-Unterprogrammen, mit der Aufgabe, einen Auftrag abzusetzen.

5.1.3 Aufrufe von Ziel-Unterprogrammen, mit der Aufgabe, das Eintreten eines Ereignisses abzuwarten (Dijkstra's P-Operation).

Ein Auftrag ist dabei eine Informationsmenge, mit deren Übergabe der Start eines neuen sequentiellen Prozesses verbunden ist. Sollte der zu beauftragende S-Bearbeiter bereits beschäftigt sein, so kann im Augenblick kein neuer Prozeß gestartet werden: Der Auftrag wird in eine Warteschlange eingereiht. Das Start-Unterprogramm kehrt im allgemeinen zurück, ohne den eigentlichen

./. 23

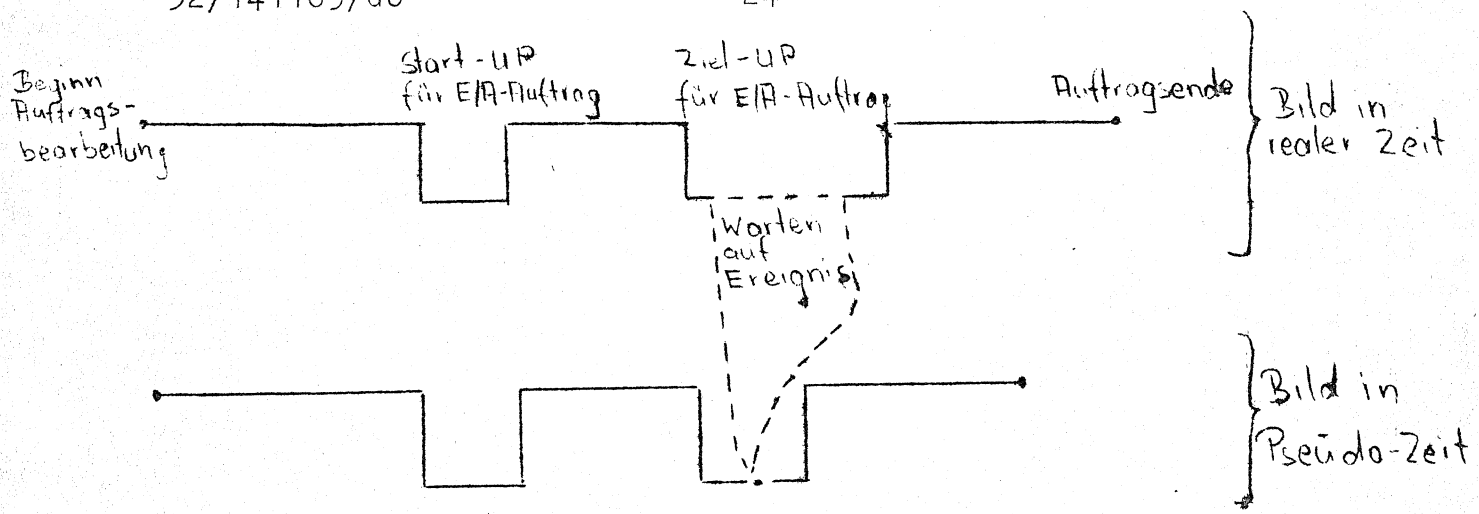
Start des Prozesses abzuwarten. Für den bereits laufenden Prozeß ergibt dann die Verpflichtung unmittelbar vor Abschluß zu prüfen, ob die Warteschlange leer ist, und eventuell den nächsten Prozeß zu starten. (Analogie: Neustart- und Fortstart-Befehle an einen E/A-Kanal.)

Ziel-Unterprogramme erlauben es, den Zustand eines anderen sequentiellen Prozesses abzufragen. Es sei ausdrücklich darauf hingewiesen, daß die Beschaffung von Zustandsinformation nur unter Zuhilfenahme von Ziel-Unterprogrammen möglich ist. Infolge der parallelen bzw. quasi-parallelen Ausführung von Prozessen ist eine Operation "Beschaffe augenblicklichen Wert einer Variablen, sperre die Variable aber nicht gegen Veränderung durch einen anderen Prozeß" bestenfalls geeignet, um die Neugierde zu befriedigen; eine Entscheidung kann nicht auf einem so gewonnenen Wert basieren. Die Sperre kann sich natürlich auch aus dem logischen Zusammenhang ergeben und muß nicht immer explizit erfolgen. Lautet die Frage etwa "Ist sequentieller Prozeß beendigt?", so kann die Antwort "ja" selbstverständlich nicht mehr verändert, d.h. zurückgenommen werden. Das ist der Grund, weshalb in 5.0.3.4 nicht explizit auf die Sperre eingegangen wurde.

5.1.4 Neben diesen Möglichkeiten gibt es natürlich noch den einfachsten Fall der Kommunikation zwischen Prozessen, nämlich, daß diese Datensätze gemeinsam benutzen. In diesem Fall sind keine generellen Regeln angebbar, außer daß auch hier die oben diskutierte Frage der Veränderungssperre geklärt werden muß.

Die Möglichkeiten 5.1.1 - 5.1.4 reichen theoretisch aus, um die Bedürfnisse der Kommunikation zwischen Prozessen zu befriedigen. Natürlich bedarf der Begriff des Ereignisses noch weiterer Untersuchungen. In praxi ergeben sich jedoch häufig S-Bearbeiter, auf denen Prozesse folgender Bauart ablaufen:

./. 24

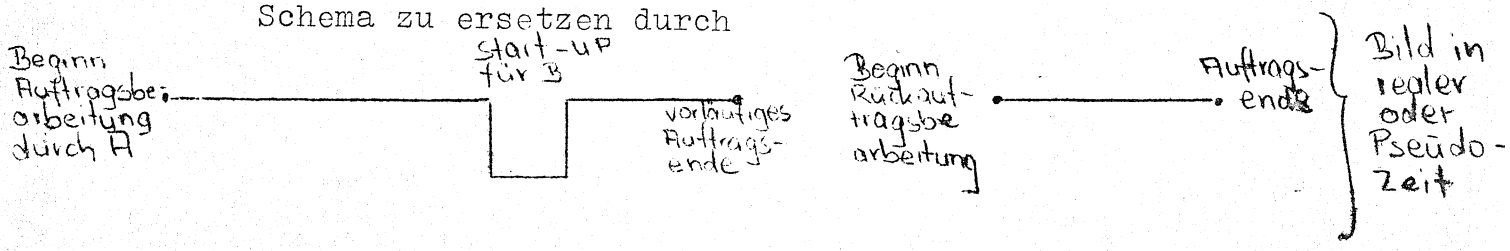


Die Wartezeit kann dabei um Größenordnungen länger sein als die eigentliche Bearbeitungsdauer. Unter bestimmten Umständen könnte der S-Bearbeiter während der Wartezeit frei für die Bearbeitung weiterer Aufträge sein. Unsere bisherige Konstruktion erlaubt jedoch nicht, dies auszunutzen. Sie verlangt stattdessen, daß weitere, mit dem gegebenenen identische S-Bearbeiter definiert werden, welche bei geeigneten Synchronisiermaßnahmen die Wartezeit zur Bearbeitung der sonst noch anstehenden Aufträge nutzen können. Diese S-Bearbeiter haben also eine gemeinsame Warteschlange für Aufträge. Diese Idee führt zu einer beträchtlichen Erhöhung der Anzahl der S-Bearbeiter, was aus speichertechnischen und organisatorischen Gründen untragbar ist. Da einige wichtige S-Bearbeiter, wie z.B. die Kernspeicherverwaltung nach obigem Modell arbeiten, ist auch die andere Möglichkeit, die Aufträge ohne Ausnutzung der Wartezeit nacheinander zu bearbeiten, unsinnig. Das System würde zu langsam werden. Wir führen daher die Hilfskonstruktion des Rückauftrags ein:

5.1.5 Ein Rückauftrag ist ein Auftrag, gegeben von einem S-Bearbeiter B an einen S-Bearbeiter A, der A mitteilt, daß ein bestimmtes Ereignis eingetreten ist. Der Rückauftrag ersetzt einen an sich von A zu gebenden Aufruf eines Ziel-Unterprogramms.

./. 25

Rückaufträge gestatten es, das im obigen Bild angegebene Schema zu ersetzen durch



Zwischen dem vorläufigen Auftragsende und dem Beginn der Bearbeitung des Rückauftrags ist der S-Bearbeiter nun frei zur Bearbeitung anderer Aufträge. Im Extremfall kbesteht die Tätigkeit des Start-Unterprogramms für Bearbeiter A darin, das Start-Unterprogramm für B aufzurufen und dann zurückzukehren. A würde dann nur über Rückaufträge gestartet werden.

Rückaufträge werden wie normale Aufträge in die Auftragswarteschlange aufgenommen, wenn der Bearbeiter im Augenblick beschäftigt ist. Da Rückaufträge erwartete Aufträge sind, kann und darf es nicht vorkommen, daß die Übernahme eines Rückauftrags in die Warteschlange, z.B. wegen Speicherschwierigkeiten, verzögert wird. Aus der in 5.2 skizzierten Einordnung der Rückaufträge wird ersichtlich, daß solche Verzögerungen zu logischen Fehlern bei der System-Konstruktion führen können. Fehler können auch auftreten, wenn Rückaufträge zu anderen Zwecken eingesetzt werden als dem, den Aufruf von Ziel-Unterprogrammen zu vermeiden.

5.2 Hierarchische Gliederung des Systems

5.2.1 Tödliche Umklammerungen

Arbeiten mehrere sequentielle Prozesse A, B, C ... zusammen, so kann infolge Konstruktionsfehlers folgende Situation eintreten: Prozeß A beauftragt Prozeß B und wartet auf das Ende des Auftrags. Zur Erfüllung des Auftrags gibt Prozeß B einen weiteren Auftrag an dem Prozeß A zugrundeliegenden Bearbeiter ab. Der letztere Auftrag kann nicht bearbeitet werden, da Prozeß A damit beschäftigt ist, zu warten. Diese als "tödliche Umklammerung" (deadly embrace, deadlock) bezeichnete Situation muß vermieden werden. Sie kann höchstens dann akzeptiert werden, wenn die statistische Wahrscheinlichkeit ihres Eintretens hinreichend niedrig ist (z.B. nur bei Zusammentreffen mehrerer voneinander unabhängiger Hardware-Fehler), und der Aufwand zur Beseitigung der Umklammerung in keinem Verhältnis zur erwarteten Häufigkeit steht. Die geforderte Mindestbedingung lautet also: Es müssen alle Situationen explizit bekannt sein, in denen eine tödliche Umklammerung möglich ist. In Wirklichkeit ist auch dies Forderung nicht erfüllbar. Es wird immer unentdeckte Programmierfehler geben, welche zu tödlichen Umklammerungen führen können.[1] Die Forderung besagt nun, daß es sich hierbei nur um lokal-beseitigbare Programmierfehler handeln dürfe, nicht um Fehler, deren Beseitigung eine Neukonstruktion von Teilen des Systems erfordern.

5.2.2 Schichten

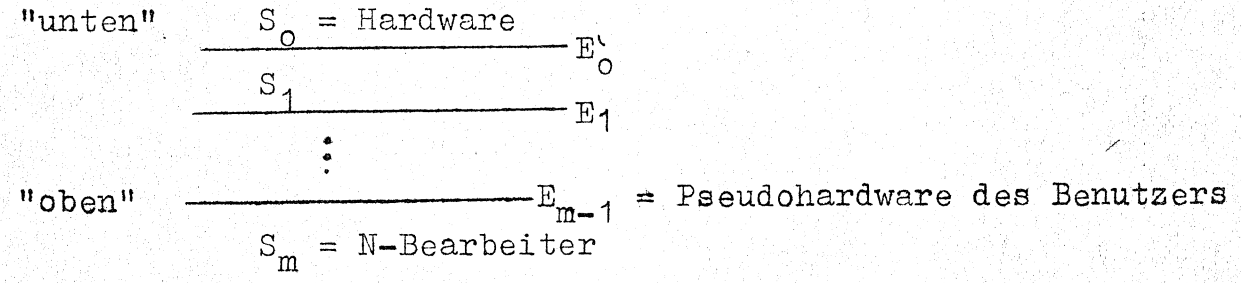
Betrachtet man das Gesamt-System einschließlich der N-Bearbeiter als eine ungeordnete Kollektion von Bearbeitern, bzw., dynamisch gesehen, als eine Kollektion von sequentiellen Prozessen, so ist es unmöglich, die obige Forderung einzuhalten. Die möglichen Auf

---

1) Es ist klar, daß es außerdem erlaubte Eingriffe seitens eines Systemprogrammierers oder Operateurs geben kann, bei denen verlangt wird, daß der sie Ausführende mit absoluter Zuverlässigkeit handelt, weil andernfalls ein Systemzusammenbruch resultiert.

tragsbeziehungensind so komplex, daß es im Entwurfsstadium überaus schwierig ist, festzustellen, ob tödliche Umklammerungen möglich sind. Dem Beispiel Dijkstras folgend helfen wir uns, indem wir die Bearbeiter und freien Unterprogramme in Gruppen S_1, ..., S_m zusammenfassen, derart, daß ein Programm der Gruppe S_i nur freie Unterprogramme der Gruppen S_j, $j<i$, aufrufen kann; in der umgekehrten Richtung $S_j \rightarrow S_i$ mit $j<i$ sind nur Rückaufträge möglich. Wir erreichen damit eine Schichtung des gesamten Systems, weshalb wir die Gruppem im weiteren als Schichten bezeichnen: Schicht S_1 kann nur Funktionen der Hardware benutzen, Schicht S_2 kann Aufträge an die Schicht S_1 geben und außerdem Hardware-Funktionen benutzen, usw. Die Komplexität des Gesamtsystems wird dahingehend reduziert, daß tödliche Umarmungen nur mehr innerhalb einer Schicht möglich sind. Die sequentiellen Prozesse einer Schicht bilden daher nun "cooperating sequential processes" im eigentlichen Sinne.

Zwischen den Schichten S_i und S_{i+1} befindet sich eine Schnittstellenebene E_i, definiert durch die Menge von Systemdiensten aus den Schichten S_1, ..., S_i, welche aus Schicht S_{i+1} aufrufbar sind. Dazu gehören auf jeden Fall alle Systemdienste, welche durch die Schicht S_i realisiert sind. Die Dienste, welche durch darunterliegende Schichten S_j realisiert sind, gehören nicht sämtlich zur Ebene E_i, sondern dienen teilweise nur der Realisierbarkeit von Systemdiensten der Schicht S_j. Die Ebene E_i definiert daher eine Pseudohardware, auf welcher Schicht S_{i+1} läuft, und für welche gewisse Betriebsmittel und Systemdienste maskiert, d.h. nicht aufrufbar, oder gar nicht vorhanden sind (wenn sie durch eine Schicht S_j, $j \geq i+1$, realisiert sind). Die oberste Ebene E_{m-1} definiert die dem Benutzer zur Verfügung gestellte Pseudohardware:

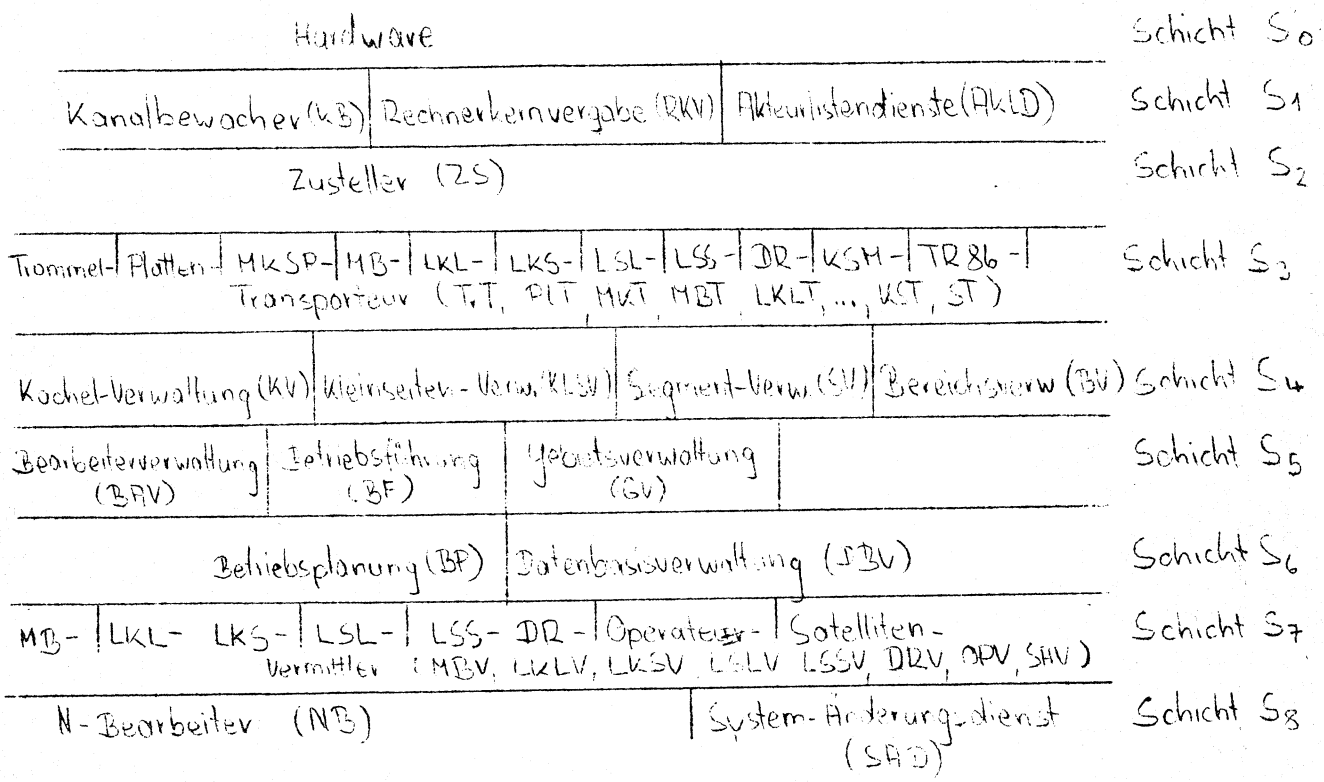


./. 28

Die Einteilung in Schichten und die Definition von Schnittstellenebenen stellt eine freiwillige Selbstbeschränkung der an der Konstruktion Beteiligten dar. Sie dient der Reduzierung der Komplexität und erzwingt eine klare Definition nicht nur der Pseudohardware des Benutzers, sondern der Schnittstelle zwischen verschiedenen Schichten. Die Einteilung ist logischer und nicht programmtechnischer Art. Sie soll sich zwar soweit wie möglich in den Programmen widerspiegeln, jedoch ist dies kein absolutes Maß: Die Eingriffsvorbehandlung gehört programmtechnisch zwar zur Schicht S_1. Wird jedoch ein Anrufseingriff entschlüsselt und sofort an Schicht S_j, $i>1$, weitergereicht, ist die Eingriffsvorbehandlung in diesem Zusammenhang als logischer Bestandteil der Schicht S_i anzusehen. Abgesehen von solchen sehr diffizilen Fällen ist jedoch darauf zu achten, daß auch programmtechnisch aus einer Schicht S_i Aufträge, Meldungen, Aufrufe usw. nicht in Schicht S_j, $j>i$, erfolgen. Bereits das erste Vorkommnis dieser Art bedeutet logisch die Zusammenfassung der Schichten S_i, S_{i+1}, ..., S_j zu einer neuen Schicht.

6. Aufbau des BSM

Für das Betriebssystem München wird folgender funktioneller Aufbau vorgeschlagen



Bemerkungen:

6.0.1 Die einzelnen Begriffe werden in 6.1 - 6.8 kurz erklärt. Die Abschnitte nehmen teilweise auf den Abschnitt 7 Bezug.

6.0.2 Die Einordnung mancher Systemteile in bestimmte Schichten ist bis zu einem gewissen Grad willkürlich.

6.0.3 Welche Systemteile durch S-Bearbeiter realisiert werden, steht insgesamt noch nicht fest. Bisher bekannte S-Bearbeite ZS, KV, BV, BF, BP, DBV, SÄD und die sämtlichen Vermittler.

6.0.4 Sollte sich ergeben, daß die Transporteure ohne Rückgriff auf den Zusteller konstruiert werden können, so wäre eine Vertauschung der Schichten 2 und 3 wünschenswert.

6.0.5 Fragen der Leistungs-Messung, Fehlerbehandlung, die Verdrängung von Gebieten von der Platte auf Magnetband, sowie eine eventuelle automatische Datensicherung durch Kopieren sämtlicher veränderter Gebiete auf MB sind noch nicht berück sichtigt.

./. 30