RECHENZENTRUM TH MÜNCHEN
ARBEITSGRUPPE BETRIEBSSYSTEME

INTERNSCHRIFT Nr. 12

THEMA:

Multics (multiplexed information and computing service)

VERFASSER:
GOOS / WOLF

DATUM:
WS 68/69

FORM DER ABFASSUNG

- ENTWURF
- [X] AUSARBEITUNG
- ENDFORM

SACHLICHE VERBINDLICHKEIT

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

ÄNDERUNGSZUSTAND

BEZUG AUF BISHERIGE INTERNSCHRIFTEN
Vorkenntnisse aus:
Erweiterung von:
Ersatz für:

BEZUG AUF KÜNFTIGE INTERNSCHRIFTEN
Vorkenntnisse zu:
Erweiterung in:
Ersetzt durch:

ANDERWEITIGE LITERATUR

Lagally

C. Multics (multiplexed information and computing service)
(Goos / Wolf)

I. Einführung: hardware, Prozesse, Adressiertechniken

I. 1 hardware

Das Betriebssystem Multics ist geplant für die Rechenanlage GE 645. Die GE 645 ist eine Spezialentwicklung aus der GE 635, die auf dieses Betriebssystem zugeschnitten ist. Das Maschinenwort der GE 645 hat 36 Bit.

Eine mögliche Konfiguration, auf der Multics laufen könnte, kann bestehen aus einer oder mehreren CPU's, Kernspeicher (mindestens 128K), Trommel(n), Plattenspeicher, Magnetbandgeräten, Lochkartenlesern, Lochkartenstanzern, Drucker usw; außerdem kann eine Anzahl von terminals (Fernschreibern) angeschlossen werden. Für real-time-Anwendungen und für den Anschluß von Bildsichtgeräten ist die Zwischenschaltung eines Satellitenrechners erforderlich.

Hervorzuheben (gegenüber anderen, insbesondere älteren Maschinen) ist die Tatsache, daß die Steuereinheiten für Hintergrundspeicher und sonstige E/A-Geräte unabhängig von der CPU arbeiten. Weiterhin sind die Zugriffszeiten und Übertragungsgeschwindigkeiten der Trommeln wesentlich beim System-design, da davon die Arbeitsgeschwindigkeit des Gesamtsystems abhängt.

I,2 Programme, Segmente, Prozesse

In den Speichermedien befinden sich "Daten". Die Einheiten der Datenmenge sind, vom Betriebssystem aus gesehen, "Segmente", d.h. Blöcke vom zusammengehörigen Worten. Ein Segment hat gewisse Eigenschaften; darunter kann die Eigenschft "ausführbar" sein. Ein ausführbares Segment heißt eine "Prozedur". Prozeduren sind prinzipiell eingriffsinvariant. Eine Prozedur enthält daher keine veränderbaren Größen. Eine Prozedur befindet sich in einem von zwei Zuständen:

(1) Sie ist inaktiv.
(2) Sie ist (ein oder mehrere Male) tätig.

Eine Prozedur, zusammen mit den Prozeduren, die von ihr explizit

aufgerufen werden, heißt ein "Programm".

Der Ablauf eines Programms heißt ein "Prozeß". Ein solcher Prozeß kann Unterprozesse haben, z.B. dann, wenn das Programm implizit, d.h. ohne es zu wissen, eine andere Prozedur startet. Ein Prozeß befindet sich in einem von drei Zuständen:

(a) Er läuft (d.h. er hat CPU-Regie).

(b) Er ist rechenfähig (d.h. er könnte sofort die CPU-Regie übernehmen).

(c) Er ist blockiert (d.h. er wartet auf die Endemeldung von E/A-Aufträgen).

Zwischen diesen Zuständen sind folgende Übergänge möglich:

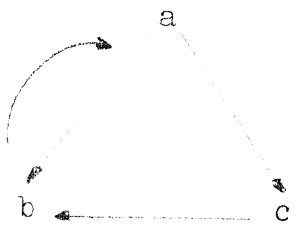

Fig. 1

Ein Prozeß kann nicht unmittelbar von Zustand "blockiert" in den Zustand "laufend" übergehen, da blockierte Prozesse bei der Regievergabe nicht berücksichtigt werden. Ein Prozeß endet mit seiner Fertigmeldung, sofern nicht durch einen anderen Prozeß ein vorzeitiges Prozeßende erzwungen wird (z.B. bei Überschreiten der Zeitschranke).

Wie bereits erwähnt, bezeichnet man als Segment (region) einen Block von zusammengehörigen Worten. Ein Segment kann maximal 2^{18} Worte haben (Ausnahme siehe unten). Falls ein Segment länger als 2^{10} Worte ist, wird es in Seiten unterteilt. Im Gegensatz zur Einteilung in Segmente ist die Einteilung eines Segments in Seiten eine rein technische Angelegenheit; denn Programmierer ist die Aufteilung seines Programms in einzelne Seiten nicht bekannt. Die Seitenlänge beträgt 1024 Worte. (Für spezielle Zwecke wird eine Seitenlänge von 64 Worten benützt). Der Hauptspeicher der Maschine enthält "Kacheln", die jeweils genau eine Seite fassen. Es wird angestrebt, die "laufend benötigtan" Seiten im Kernspeicher zu halten (siehe Bemerkung am Schluß vom I, 3).

Prinzipiell sind auch Segmente mit mehr als 2^{18} Worten zugelassen. Solche unbeschränkten Segmente können mit enem oder mehreren "Fenstern", d.h. Segmenten, die höchstens 2^{18} Worte groß sind, betrachtet werden.

Für die in der Rechenanlage laufenden Prozesse sind die folgenden speziellen files vorhanden (wegen des synonymen Gebrauchs der Begriffe "file" und "segment" vgl. II,1):

monitory file: ein Datensegment, das Informationen, z.B. Fehlermeldungen über den laufenden Prozeß enthält. Es kann nur vom System beschrieben werden, darf jedoch nach beendetem Lauf vom Prozeß zur Fehlersuche benutzt werden. Ein solches Segment wird vom System über jeden laufenden Prozeß angelegt.

standard monitory file: ein Datensegment, das alle Informationen aufnimmt, die das System für sich selbst für nötig hält. Es ist den Prozessen nicht zugänglich, vielmehr dient es zum Beispiel zur Information des Operateurs bei Systemzusammenbrüchen. Es ist nur einmal vorhanden.

stack file: ein Datensegment, das der Prozeß verwendet, um temporäre Daten zu speichern (d.h. der stack file enthält den eigentlichen Arbeitsspeicher). Bei jedem "Prozeduraufruf" während des Prozesses wird im stack file Speicher für die neue Prozedur neu belegt.

geheimer stack file: In diesem Segment wird beim Start eines Unterprozesses alle die Information aufgenommen, die bei Wiederaufnahme des Oberprozesses von Bedeutung ist, z.B. sind zu diesem Zeitpunkt die Registerinhalte des Oberprozesses sicherzustellen. Bei einer Unterbrechung des Unterprozesses dürfen diese Registerinhalte nicht von den entsprechenden Inhalten des Unterprozesses überschrieben werden, weshalb für den Unterprozess ebenfalls Speicher im geheimen stack file reserviert ist. Dieser file ist

dem Programmierer nicht zugänglich.

Verweissegment: In diesem Segment wird bei Aufruf einer "Prozedur" Speicher reserviert, der die Informationen aufnimmt, die an sich zur Prozedur gehören, aber deswegen, weil sie variabel sind, nicht im Prozedursegment gespeichert werden können (wegen Eintrittsinvarianz).

I, 3 Adressiertecheniken

Jedes Wort eines Segments wird durch eine "verallgemeinerte Adresse" identifiziert. Wie Fig.2 zeigt, besteht eine solche verallgemeinerte Adresse aus zwei Teilen:

1. der Segmentangabe (= Nummer des Segments)
2. der Segmentadresse (= Adresse des Wortes innerhalb des Segments).

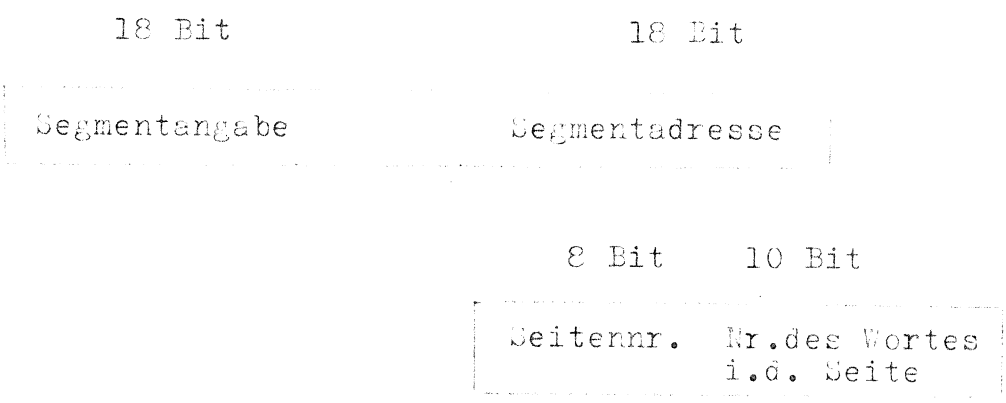

Fig. 2

Die Segmentadresse läßt sich aufteilen in die Seitennummer und in die Nummer des Wortes innerhalb der Seite. Das Adressenrechenwerk des Prozessors ist so gebaut, daß der Prozess mittels der verallgemeinerten Adresse auf jedes Wort zugreifen kann, das im Kernspeicher steht. Zusammen mit der software des Betriebssystems, die für eventuell erforderliche Transporte in dem Hauptspeicher sorgt, kann der Prozeß mittels der verallgemeinerten Adresse jedes Wort ansprechen, gleich wo es in der Speicherhierarhie liegt.

Will man auf ein Wort innerhalb einer Seite zugreifen, so muß

bekannt sein, ob und wo diese Seite im Kernspeicher liegt. Dazu dient die "Segmentseitentafel", die für jedes Segment vorhanden ist. In dieser Segmentseitentafel sind für die im Kernspeicher stehenden Seiten die absoluten Adressen eingetragen.

Um mit verallgemeinerten Adressen arbeiten zu können, enthält jeder Prozessor der Rechenanlage neben dem Akkumulator, dem Multiplikator-Quotienten-Register, acht Indexregistern X0,....X7 und dem Befehlszähler, die den üblichen Zwecken dienen, weitere Register: ein "Deskriptorbasisregister", ein "Prozedurbasisregister" und vier doppeltlange "Basisregister". Auf die Funktion des Deskriptorbasisregisters wird später eingegangen. Jedes der vier Basisregister enthält eine verallgemeinerte Adresse (Segmentangabe/Segmentadresse) und die zugehörige physikalische Adresse, im allgemeinen die Anfangsadresse eines Segments (falls dieses Segment nicht in Seiten eingeteilt ist) bzw. die Adresse der Segmentseitentafel. Die vier Basisregister sind gemäß ihrer speziellen Funktion in Multics benannt:

Basisregister Nr.	Aufgabe
0	argument pointer
1	base pointer
2	linkage pointer
3	stack pointer

Der "argument pointer" weist auf Versorgungsdaten für den Prozeß. In den "base pointer" kann die verallgemeinerte Adresse eines beliebigen Segments geladen werden. Der "linkage pointer" enthält die verallgemeinerte Adresse des Verweissegments. Der "stack pointer" zeigt auf den Anfang des Arbeitsspeichers für den laufenden Prozeß. Das Prozedurbasisregister enthält die Segmentangabe (Segmentnummer) des laufenden Prozedursegments.

Folgende Adressierungstechniken sind möglich:

1) Direkte Adressierung

Die verallgemeinerte Adresse wird ausgehend vom Adressteil eines Befehlswortes gebildet. Diese Form gestattet es, auf Worte innerhalb des Segment, das gerade ausgeführt wird, oder auf Worte innerhalb eines der vier Segmente, deren Segmentangabe in dem Basisregistern steht, zuzugreifen.

Im zweiten Fall ist ein spezielles Bit (external flag) innerhalb des Befehlswortes gesetzt. Dann wird durch ein Feld am Anfang des Befehlswortes (segment tag) spezifiziert, welches Basisregister angesprochen wird. Aus diesem Basisregister ist die verallgemeinerte Adresse zu nehmen. Der zweiter Teil, die Segmentadresse, wird mit der im Befehl enthaltenen Adresse und gegebenfalls nach einem im Befehlswort angegebenen Modus mit einem der Indexregister modifiziert.

Im ersten Fall ist das entsprechende Bit nicht gesetzt. Die verallgemeinerte Adresse wird aus dem Inhalt des Prozedurbasisregisters (= Segmentangabe des laufenden Prozesses),dem Inhalt des Befehlszählers (Segmentadresse) und dem Adreßteil des Befehls zusammengesetzt.

2) Indirekte Adressierung ITS (indirect to segment)

In diesem Fall wird die wie unter 1) gebildete verallgemeinerte Adresse benutzt, um aus einem Segment, (etwa dem Verweissegment) zwei Worte zu je 36 Bit zu holen. Das erste Wort enthält eine Segmentangabe und den Hinweis, daß die Adressierung " indirect to segment" erfolgt. Das zweite enthält die Segmentadresse und eine Modusangabe. Aus Segmentangabe und Segmentadresse wird eine verallgemeinerte Adresse gebildet, wobei gegebenenfalls noch entsprechend der Modusangabe mittels eines Indexspeichers modifiziert wird.

3) Indirekte Adressierung ITB (indirekt through base)

Wie unter 2) werden zwei Worte geholt. Im Unterschied zu 2) enthält das erste Wort den Hinweis, daß die Adressierung "indirect through base" erfolgt und eine Zahl zwischen 0 und 3, die angibt, welches Basisregister benutzt werden soll. Dann wird als erster Teil der verallgemeinerten Adresse die Segmentangabe aus dem betreffenden Basisregister verwendet, im übrigen wird wie unter 2) vorgegangen.

4) Indirekte Adressierung über file-Namen

Da eine Prozedur invariant gegenüber der Neukompilierung anderer Segmente sein soll, kann ein Prozedursegment ein fremdes Segment zunächst nur über einen file-Namen ansprechen. Diese Art der Adressierung wird nun beim ersten Zugriff auf dieses Segment

auf 2) zurückgeführt.

Für jeden laufenden Prozeß wird eine "Segmentnamenstafel" (descriptor segment) angelegt, in der allen dem Prozeß "bekannten" Segmenten (d.h. solchen, auf die er bereits zugegriffen hat) fortlaufende Nummern zugeordnet werden. Diese Segmentnummer (Segmentangabe) gibt also die Position des zu dem betreffenden Segment gehörigen Eintrags innerhalb der Segmentnamenstafel an. Da für jeden Prozeß eine eigene Segmentnamenstafel angelegt wird, können zu einem Segment verschiedene Segmentnummern(in verschiedenen Tafeln) gehören. Beim Aufruf eines Segments durch einen Namen wird die Segmentnamenstafel nach diesem Namen durchsucht. Ist kein entsprechender Eintrag vorhanden, so ist das Segment dem Prozeß noch nicht bekannt. Dann wird zunächst der Segmentname in einen Segnamen umgewandelt, um dieses Segment innerhalb der Hierarchie des file-Systems zu lokalisieren (siehe II,1). Dem Segment wird eine Segmentnummer zugeordnet und diese Angabe wird in die Segmentnamenstafel eingetragen. Da die Informationen über die Verbindung zwischen dem laufenden Prozeß und dem angesprochenen Segment nicht in das Prozedursegment geschrieben werden dürfen (Eine Prozedur darf keine variablen Größen enthalten!), wird hierfür ein eigenes Segment, das Verweissegment (vgl. I,2) benutzt. Der Zugriff erfolgt dann in allen folgenden Aufrufen des Segments wie unter 2) beschrieben.

Es bleibt noch auszuführen, wie man aus der verallgemeinerten Adresse die zugehörige physikalische Adresse erhält:

Hat man die verallgemeinerte Adresse, so muß man feststellen, ob und wo die zugehörige Seite im Kernspeicher liegt. Dazu ist, wie erwähnt, die Seitentafel des Segments nötig. Ob die Segmentseitentafel des angesprochenen Segments im Kernspeicher liegt und wo, entnimmt man der (systemuniversellen) "Segmentstatustafel". Liegt zu einem Segment die Seitentafel im Kernspeicher, so heißt dieses Segment "aktiv", sonst "inaktiv". Den Zugriff zu einem Wort im Segment Nr. i erläutert Fig. 3.

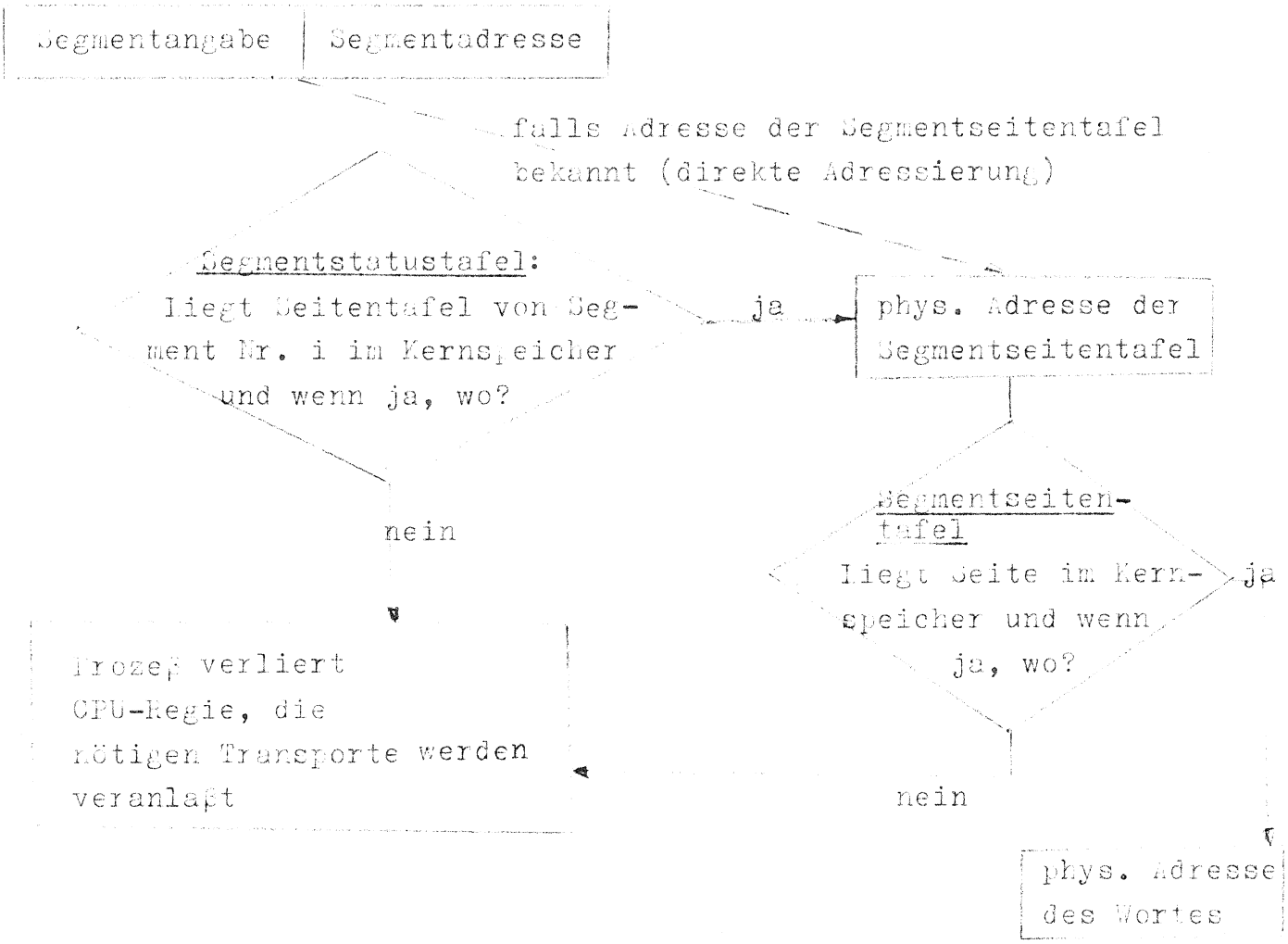

Fig. 3

Um die für diesen Vorgang im allgemeinen nötigen zwei Kernspeicherzugriffe in möglichst vielen Fällen zu vermeiden, sind vier Assoziativregister vorgesehen. Zu jedem dieser Register steht der aus Segmentangabe und Seitennummer bestehende Teil einer verallgemeinerten Adresse (18+8 Bit). Die vier gespeicherten Teile von verallgemeinerten Adressen weisen auf vier verschiedene im Kernspeicher befindliche Seiten. Mit jedem Assoziativregister ist ein Register verbunden, das die Kernspeicheradresse der betreffenden Seite (als Vielfaches um 1024, also der Seitenlänge) enthält. Wird eine verallgemeinerte Adresse angegeben, so wird zunächst (gleichzeitig) in den 4 Assoziativregistern nachgesehen, ob die gegebene Segmentangabe und Seitennummer mit dem Inhalt eines dieser Register übereinstimmt; trifft dies zu, so wird aus dem mit dem betreffenden Assoziativregister verbundenen Register die zugehörige absolute Adresse geliefert. Andernfalls wird der Zugriff auf dem in Fig. 3 angegebenen Weg versucht.

Wenn eine verlangte Segmentangabe und Seitennummer nicht in einem der Assoziativregister zu finden ist, die betreffende Seite aber im Kernspeicher steht, so wird nicht nur die gewünschte absolute Adresse geliefert, sondern es wird darüber hinaus angenommen, daß weitere Zugriffe auf diese Seite erfolgen. Deshalb werden die entsprechenden Angaben für diese Seite in eines der Assoziativregister geladen, wodurch der vorherige Inhalt dieses Assoziativregisters zerstört wird. Nach Möglichkeit soll ein solches Register überschrieben werden, dessen Inhalt voraussichtlich nicht häufig gebraucht werden wird. Als Kriterium hierfür wird angesehen, wie oft eine solche Adresse benutzt wurde, was zu diesem Zweck in den Assoziativregistern gezählt wird.

Informationen ähnlicher Art benötigt das System, um zu entscheiden, welche Seiten im Kernspeicher gehalten werden sollen. Dazu ist von Interesse, ob eine Seite überhaupt benutzt wurde, was durch ein "use bit" in der Seitentafel (und im Assoziativregister) angegeben wird. Analog wird bei Veränderung einer Seite (z.B. Schreiben) ein "Änderungsbit" in der Seitentafel gesetzt.

II. Das file - System

II,1 Struktur des file-Systems

Zur Aufbewahrung der Information dienen files. Die Begriffe Segment und file sind gleichbedeutend. Zu Anfang werden files mit Hilfe von Namen angesprochen. Um einen file zu finden, muß dem System der Name des files in eindeutiger Form bekannt sein. Zusätzlich muß eine Zuordnungstabelle (file-Name ⟷ physikalische Adresse) vorhanden sein. Um Namenskollisionen zu vermeiden, wird die Zuordnungstabelle in Form eines Baumes aufgebaut, dessen Wurzel dem System bekannt ist. Diese baumförmige Struktur besteht aus mehreren "Verweistabellen" (file directories). Jede Verweistabelle ist selbst ein file und hat einen Namen. Die Elemente ("Einträge") der Verweistabellen sind Angaben über andere files, speziell andere Verweistabelle.

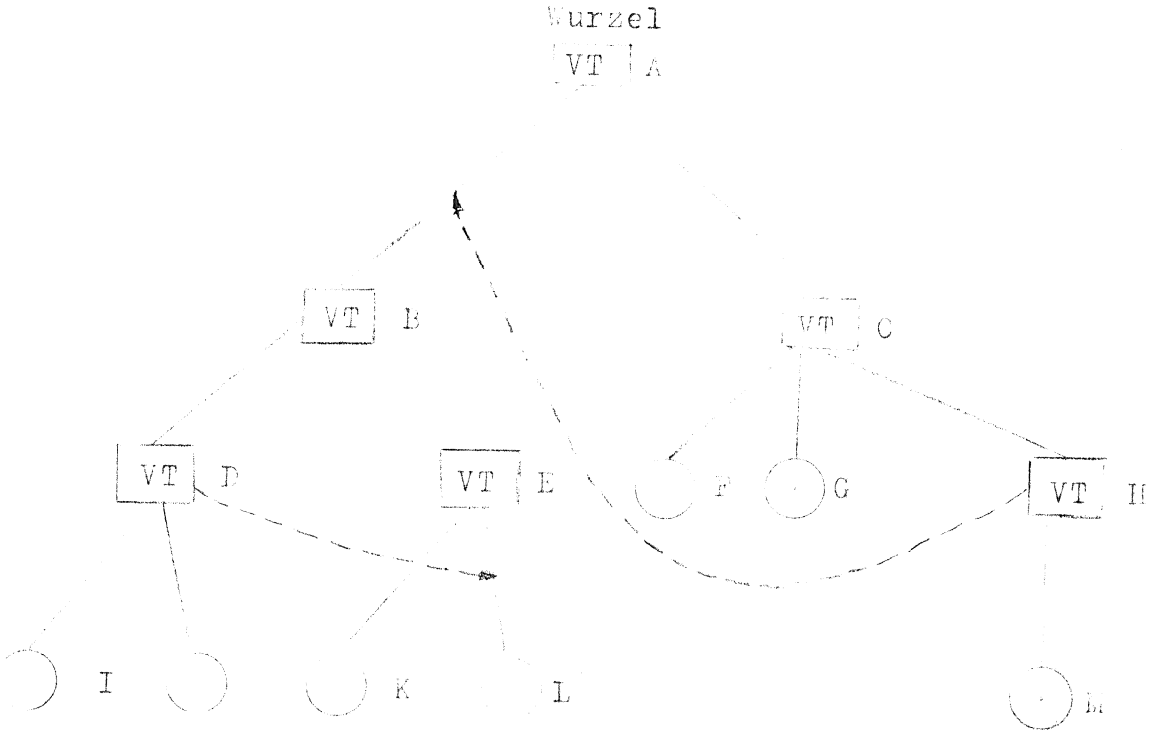

Fig. 4

Zwei Typen von solchen Einträgen sind möglich:

1) "Verweiseinträge" (branches), die einem file-Namen eine physikalische Adresse zuordnen (in der Fig. 4 ausgezogene linien);

2) "Hinweiseinträge" (links); die einem file-Namen einen Eintrag in einer anderen Verweistabelle zuordnen (in der Fig. 4 gestrichelte Linien; die Pfeilspitze zeigt eigentlich auf den zu diesem file gehörigen Eintrag in der übergeordneten Verweistabelle.

Files können folgendermaßen identifiziert werden:

a) durch einen file-Namen relativ zur direkt übergeordneten Verweistabelle;
b) durch einen file-Namen relativ zu einer beliebigen übergeordneten Verweistabelle;
c) durch einen absoluten file-Namen (tree name);
d) durch einen Wegnamen (path name);
e) durch einen allgemeinen Wegnamen.

ad a) Ein file hat einen Namen, der zunächst nur festgelegt ist relativ zur Verweistabelle des Prozesses, der ihn aufgebaut hat, der "Arbeitsverweistabelle" (working directory). Der file-Name muß bezüglich dieser direkt übergeordneten Verweistabelle eindeutig sein; eine andere Verweistabelle kann einen direkt untergeordneten file gleichen Namens besitzen.

ad b) Aus einem file-Namen relativ zur direkt übergeordneten Verweistabelle erhält man einen file-Namen relativ zu einer beliebigen übergeordneten Verweistabelle, indem man Schritt für Schritt im Baum aufsteigt und die Namen der entsprechenden Verweistabellen (ausgenommen diejenige, auf die der file-Name bezogen wird) davorsetzt. Relativnamen werden durch einen vorgesetzten Doppelpunkt gekennzeichnet. Z.B. hat der file J bezüglich der Verweistabelle B (vgl. Fig. 4) den relativen Namen

: D : J.

ad c) Einen absoluten file-Namen erhält man, indem man wie in b) vorgeht, jedoch bis zur Wurzel des Baums aufsteigt. Der absolute file-Name beginnt mit dem Namen der Wurzel, es wird kein Doppelpunkt vor den Namen gesetzt. Der file J (vgl. Fig. 4) hat den absoluten Namen

A : B : D : J .

ad d)Ein Wegname lokalisiert einen file f1 gegenüber einer anderen, nicht notwendig übergeordneten Verweistabelle f2. Um diesen Wegnamen zu erhalten, steigt man von f1 Schritt für Schritt im Baum auf (wobei für jeden solchen Schritt ein Stern gesetzt wird), bis man zu einer Verweistabelle gelangt, die f 1 und f 2 übergeordnet ist. Der Abstieg zu f 2 wird dann durch den relativen Namen dieses files bezüglich der übergeordneten Verweistabelle beschrieben. Der file L ist gegenüber D (vgl. Fig. 4) durch den Wegnamen

: * : * : D : L

festgelegt. Wie in b) wird vor den Namen ein Doppelpunkt gesetzt.

ad e)Ein allgemeiner Wegname wird wie unter d) gebildet, jedoch können dazu auch Hinweiseinträge benutzt werden. Ein allgemeiner Wegname z.B. für den file L bezüglich der Verweistabelle C (vgl. Fig. 4) kann folgendermaßen zustandekommen: In D existiere ein Hinweiseintrag auf L in der Form

(D1, A : B : E : I)

oder (D1, : * : B E L);

dann kann der file L gegenüber C durch den allgemeinen Wegnamen

: * : B : D : D1

lokalisiert werden.

Von den angegebenen Identifizierungsmöglichkeiten für einen file ist a) Spezialfall vom b), b) Spezialfall vom d) und d) Spezialfall vom e). Zu beachten ist jedoch, daß sich diese fünf Typen von file-Namen u.U. unterschiedlich gegenüber Änderungen in einer Verweistabelle verhalten. Man beachte, daß ein Verweisantrag stets einen Namen der Art a) erfordert. Hinweiseinträge spezifizieren einen Weg, um ausgehend von einer beliebigen Verweistabelle, den Verweiseintrag für einen bestimmten file zu finden.

II,2 Zugriffskontrolle

Versucht ein Benutzer (oder ein Prozeß, eine Gruppe von Prozessen oder Benutzern) auf einen file zuzugreifen, so muß eine Erlaubnis vorliegen, die davon abhängt, was der Benutzer mit

dem betreffenden file tun möchte.

Ein normales Segment (file, keine Verweistabelle) kann gegenüber einem Prozeß folgende Eigenschaften haben.

1) Der Prozeß darf in diesem Segment lesen (READ)
2) Er darf in diesem Segment schreiben (WRITE)
3) Er darf dieses Segment ausführen (EXECUTE).
4) Er darf zu diesem Segment Information hinzufügen (APPEND)

Diese vier Benutzungsmodi sind für Verweistabellen ebenfalls erlaubt, haben aber hier eine etwas abgeänderte Bedeutung:

1) lesen: Es können Einträge aus diesem Segment gelesen werden.
2) Schreiben: Bestehende Einträge dürfen abgeändert werden.
3) Ausführen: Unbeschränktes lesen (Durchsuchung der Verweistabelle) ist erlaubt.
4) Hinzufügen: Der Prozeß darf Neueinträge in die Verweistabelle machen.

Will ein Prozeß auf ein Segment zugreifen, so muß er durch irgendeinen Wegnamen zunächst an den Verweiseintrag (nicht Hinweiseintrag) für dieses Segment gelangen. Dieser steht in der dem Segment (file) direkt übergeordneten Verweistabelle. Mit diesem Verweiseintrag ist eine "Zugriffskontrolliste" (access control list) verbunden. In dieser Liste sind alle diejenigen Benutzer spezifiziert, denen Zugriff zu dem eingetragenen file erlaubt ist. Zusätzlich ist angegeben, in welchen Modi dieser Zugriff erfolgen darf, und zwar dadurch, daß je 1 Bit für jeden Modus gesetzt ist (Zugang in diesem Modus erlaubt) oder nicht gesetzt ist (Zugang in diesem Modus verboten).

Für jeden Benutzer ist zusätzlich ein weiteres Bit vorhanden, das sogenannte "Unterbrechungs-Bit" (TRAP bit). Dieses Bit wird bei einem Zugriff als erstes abgefragt. Ist es gesetzt, so wird der versuchte Zugriff auf dieses Segment zunächst unterbrochen und ein oder mehrere vom System oder vom Benutzer spezifizierte Unterprozesse gestartet. Zu diesem Zweck kann für jeden Benutzer in der Zutriffskontrolliste eine Unterliste gehalten werden, die "Unterbrechungsliste"(trap list).

Die Unterbrechungsliste enthält Einträge für die zu startenden Prozeduren und deren Parameterversorgung. Diese Prozeduren werden in der Reihenfolge, in der sie in der Unterbrechungsliste stehen, gestartet. Sie erhalten die in dieser Liste enthaltenen Parameter mitgeteilt und zusätzlich Informationen über die Vorgeschichte, die zu ihrem Aufruf führte. Eine solche Unterbrechungsprozedur kann z.B. folgende Aufgaben erledigen:

1) Sie setzt die Zugriffsbits um (was nur erlaubt ist wenn vorher das WRITE-Bit gesetzt war).
2) Sie verlangt ein Codewort von dem Prozess, der die Unterbrechung verursacht hat. (Ein solches Codewort kann einem Prozeß mittels eines speziellen Kommandos zugeordnet werden; diese Zuordnung kann durch Kommando auch wieder aufgehoben werden. Dadurch kann die Geheimhaltung eines files garantiert werden.)
3) Sie beschafft Testinformation.
4) Das Unterbrechungsbit ist sicher gesetzt, wenn das Segment nicht in einem der on-line-Speicher (Trommel, Platte) vorhanden ist. In diesem Fall gehört zu den Parametern einer Unterbrechungsprozedur die Spezifikation der Lage auf Magnetband oder sonstigen abgesetzten Speichermedien. Die Unterbrechungsprozedur kann dem aufrufenden Prozeß mitteilen, daß der Zugriff zu diesem Segment längere Zeit benötigen würde und fragen, ob der Zugriff unter diesen Umständen noch zweckmäßig ist. Je nach Antwort kann sie dann einen anderen Prozeß starten oder dem Operateur die für das Einhängen des entsprechenden Magnetbandes nötigen Parameter angeben.
5) Mehrere Prozesse, die ein Segment benutzen dürfen, können sich über eine Unterbrechungsprozedur gegenseitig Mitteilungen machen.

II,3 Backup-System (Sekundärspeicherunterstützung)

Dem Benutzer wird der Eindruck vermittelt,daß das file-System (nahezu) unbegrenzte Kapazität besitzt.Da aber die Kapazität von Trommel und Platte begrenzt ist,müssen Segmente z.B.auf Magnetbänder geschrieben werden können.Dadurch wird der file jedoch dem direkten Zugriff des Systems entzogen.Jedoch kann der file nach wie vor in einer Verweistabelle eingetragen sein.In diesem Fall wird in der Zugriffskontrolliste das Unterbrechungsbit gesetzt und das Band spezifiziert,auf dem der file zu finden ist.

Weiterhin können abgesetzte Hintergrundspeicher - wobei hier der Einfachheit halber nur Magnetbänder betrachtet werden- bei teilweisem oder vollständigem Informationsverlust auf Trommel und Platte zur Durchführung von Hilfsmaßnahmen benutzt werden.
Diesem Zweck dienen die folgenden Vorkehrungen:

1) Bei Abmeldung eines Prozesses wird jeder durch diesen Prozeß geänderte file auf zwei Magnetbänder kopiert.
   (1 MB würde genügen,erscheint jedoch nicht sicher genug.)
2) Alle n Stunden werden alle in dieser Zeit geänderten files (wenn nicht bereits unter 1) geschehen) auf zwei Magnetbänder kopiert.
   (1) und 2) werden als "incremental dump" bezeichnet.)
3) Alle m Wochen werden die zum System gehörenden Segmente, alle Verweistabellen und die in den letzten m Wochen veränderten files auf zwei Magnetbänder kopiert ("weekly dump").

Mittels der durch diese Maßnahmen zur Verfügung stehenden Magnetbänder ist folgende Ladeprozedur etwa bei vollständigem Informationsverlust auf Trommel und Platte möglich:

Zuerst wird eine Kopie des Systems (System-files) aus Bändern des letzten weekly dump geladen.Danach kann das System gestartet werden,und der Ladeprozeß kann unter der Kontrolle des Systems fortgesetzt werden.Sollten seit dem letzten weekly dump größere Systemänderungen vorgenommen worden sein,so sind vor dem Start des Systems noch die entsprechenden Bänder aus den incremental dumps zu laden.Nachdem die Systemfiles geladen sind,werden die Bänder,die von incremental dumps stammen,in umgekehrter zeitlicher Reihenfolge geladen,und zwar bis zum Zeitpunkt des letzten weekly dump.Danach wird der zweite Teil des letzten weekly dump geladen.Während des Ladeprozesses werden überflüssige oder ver-

altete files nicht berücksichtigt.Da beim Zurückgehen zu früheren Zeitpunkten immer weniger bedeutungsvolle Information auftaucht, ist es zweckmäßig,zunächst nur einen Teil der benötigten files zu laden und bei den Einträgen für die nicht geladenen files das Unterbrechungsbit zu setzen,so daß diese files erst bei Bedarf geladen werden.Das angegebene Verfahren sollte jedoch nur in Extremfällen angewandt werden.

Für leichtere Fehler ist eine Prozedur zur Einzelkorrektur im on-line-Speicher vorgesehen.Damit können insbesondere Fehler von der Art beseitigt werden,wie sie durch einen Zusammenbruch des Systems entstehen können,daß nämlich in verschiedenen Segmenten Änderungen vorgenommen wurden,ohne daß das System die entsprechenden Eintragungen in den Verweistabellen vollständig ausführen konnte.Die Prozedur ist so angelegt,daß sie alle Verweistabellen durchliest und inkonsistente Informationen soweit wie möglich korrigiert.Verbleibende fehlerhafte Einträge oder files werden gelöscht bzw. an der Stelle abgeschnitten,an der der Fehler auftäuchte.

Die unter 1) 2) und 3) angegebenen Maßnahmen bieten Ansatzpunkte für Kritik.Insbesondere erscheint der zur Rettung von Information getriebene Aufwand zu erheblich.Kaum zu rechtfertigen ist die unter 1) angegebene Maßnahme.Es erscheint zweckmäßiger,etwa bei Abmeldung eines files abzufragen,ob eine (oder zwei) Kopien auf Magnetband nötig sind oder nicht.Weniger stichhaltig ist der gegen 2) erhobene Einwand,daß auf dem Magnetband ein inkonsistentes Bild eines Prozesses entsteht, weil der Prozeß z.B. während des Dumps noch in ein Segment schreibt.Das kann verhindert werden,indem einem Prozeß zuerst die Regie genommen wird,ehe die zugehörigen files auf Magnetband geschrieben werden.Vorsicht ist jedoch geboten bei zeitlich miteinander verketteten Prozessen (cooperating sequential processes).

II,4 Die Programmstruktur des file-Systems

Im folgenden soll an Hand von Fig.5 kurz die grundlegende Programmstruktur des file-Systems erläutert werden.Die Moduln des file-Systems haben im wesentlichen folgende Aufgaben:

1) Verweistabellen bestehender Segmente (files) zu unterhalten,
2) auf Anforderung Segmente zugänglich zu machen,
3) neue Segmente zu erzeugen,
4) vorhandene Segmente zu löschen.

Die Moduln des file-Systems können Informationen aus folgenden Segmenten bzw.Tafeln entnehmen:

1) segment name table (Segmentnamenstafel):
   Eine solche Tafel wird für jeden laufenden Prozeß angelegt und liefert für jeden dem Prozeß bekannten file die Zuordnung file-Name ⟷ Segmentangabe.
2) segment status table (Segmentstatustafel):
   Aus dieser Tafel kann entnommen werden,ob und wo sich die Seitentafel eines Segments im Kernspeicher befindet.
3) synonym table (Synonymtafel):
   Sie ermöglicht es,files unter wechselnden Namen aufzurufen.
4) directories (Verweistabellen)
5) working directory table:
   Sie enthält den absoluten Namen der Arbeitsverweistabelle des laufenden Prozesses.
6) page out table:
   In dieser Tafel werden diejenigen Seiten eingetragen, die bei Bedarf aus dem Kernspeicher verdrängt werden können.
7) active file table :
   In dieser Tafel sind die aktiven files eingetragen,sie enthält darüber hinaus Informationen,die zur Überwachung von E/A-Forderungen an diese files benötigt werden.
8) Weiterhin sind die auf E/A wartenden Prozesse in einer Warteschlange (I/O queue) aufgereiht.

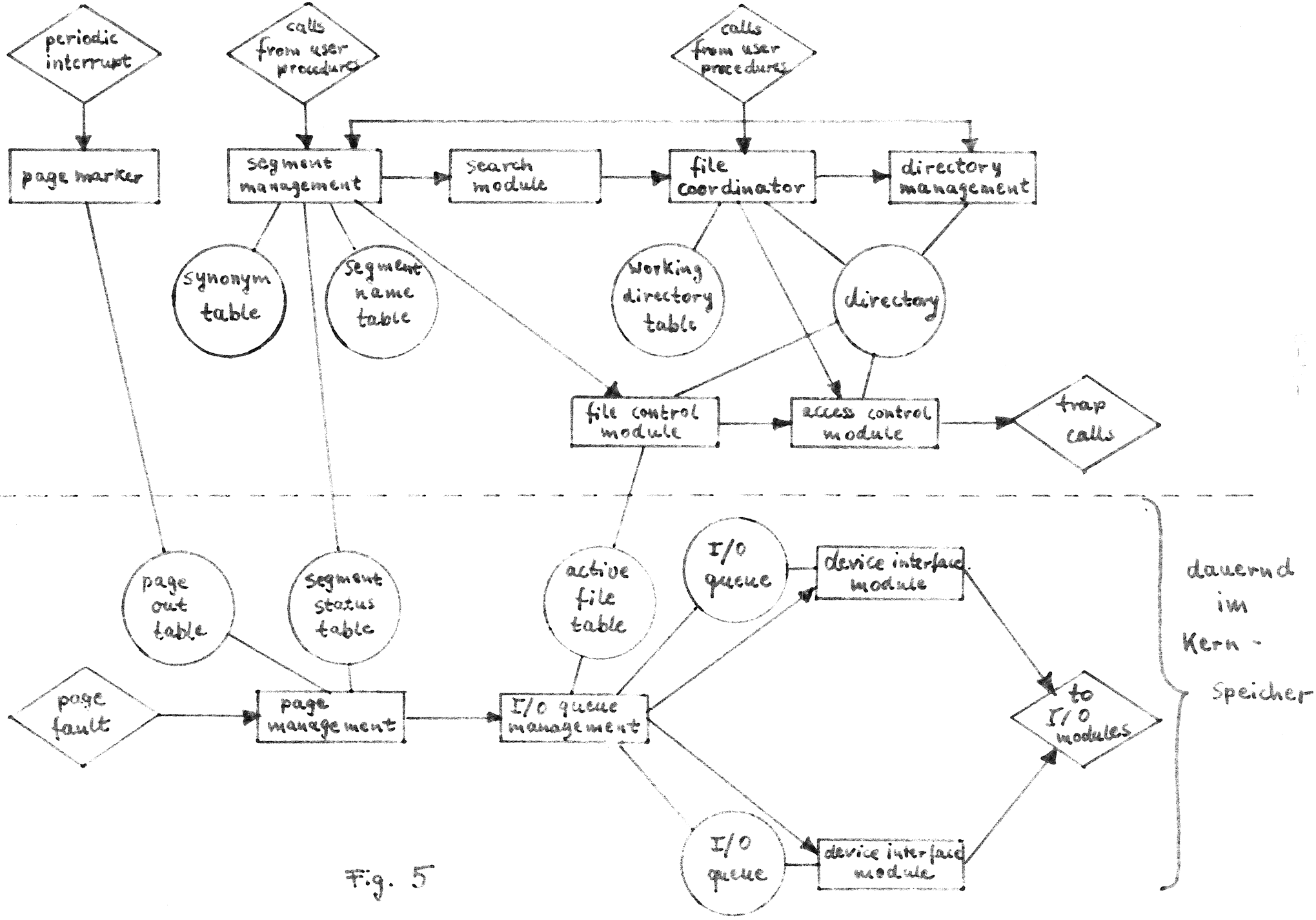
periodic interrupt
calls from user procedures
calls from user procedures
page marker
segment management
search module
file coordinator
directory management
synonym table
segment name table
working directory table
directory
file control module
access control module
trap calls
page out table
segment status table
active file table
I/O queue
device interface module
dauernd im Kern - Speicher
page fault
page management
I/O queue management
to I/O modules
I/O queue
device interface module

Fig. 5

Einige Moduln (mit ihren wesentlichen Aufgaben) sind die folgenden:

1) segment managing module:
Er tritt in Aktion,wenn ein Prozeß (Benutzer) auf ein Segment zugreift.Ist das Segment,auf das zugegriffen wird,dem Prozeß noch nicht bekannt,so veranlaßt er die folgenden Maßnahmen (die z.T. von anderen Moduln ausgeführt werden):

Die Lage dieses Segments in der Hierarchie der Verweistabellen des Benutzers wird ermittelt (siehe 2) weiter unten).Über den Verweisabschnitt (linkage section) wird, wie in I,3 beschrieben,die Verbindung zu diesem Segment hergestellt.Weiterhin werden eventuell nötige Transporte veranlaßt und die entsprechenden Neueinträge bzw. Änderungen in Segmentnamenstafel und Segmentstatustafel vorgenommen.

Bei bereits bekannten,aber nicht aktiven Segmenten wird nur ein Teil dieser Schritte durchlaufen.

2) search module:
Er wird vom segment management module aufgerufen,um ein Segment in der Hierarchie der Verweistabellen eines Benutzers zu lokalisieren.

3) file coordinator:
Er veranlaßt das Durchsuchen der Arbeitsverweistabelle und besorgt Neueinträge,Änderungen und das Löschen von Einträgen in dieser Tabelle.Weiterhin kann er mittels des access control module (siehe 6) weiter unten) Änderungen in der Zugriffskontrolliste vornehmen .Außerdem kann eine Benutzerprozedur auf dem Weg über diesen Modul eine Seite (für eine gewisse Zeit) im Kernspeicher festhalten.

4) directory management module:
Er wird etwa vom file coordinator aufgerufen,um die Arbeitsverweistabelle zu durchsuchen.

5) file control module:
Er nimmt (u.a.) auf Veranlassung des segment management module Einträge in der active file table vor.Dieser

Modul kontrolliert auch die Benutzung eines files durch verschiedene Benutzer.Dabei ist zugelassen,daß mehrere Benutzer in diesem file lesen oder daß 1 Benutzer in diesem file liest und schreibt.

6) access control module:
Dieser Modul wertet Informationen aus der Zugriffskontrolliste aus,was zum Aufruf von Unterbrechungsprozeduren (trap calls) führen kann.

7) page marker:
Er wird durch periodische Unterbrechung (Weckerprogramm) an die Regie gebracht und ermittelt auf Grund der "use bits" diejenigen Seiten im Kernspeicher,die in die page out table eingetragen werden.

8) page management:
Dieser Modul wird aufgerufen,sobald Seiten verlangt werden,die nicht im Kernspeicher stehen. Er veranlaßt, daß das I/O queue management die nötigen Transportprogramme in die Schlange der auf E/A wartenden Programme einreiht.Kommt ein solches Transportprogramm an die Reihe,so werden über einen device interface module (Geräteverwalter) die Steuereinheiten der betreffenden E/A-Geräte angesprochen.

III.Allgemeines zum Aufbau von Multics

III,1 Grundvorstellungen,Definition des Begriffes Prozeß

Ausgangspunkt für den Aufbau von Multics sind folgende Grundvorstellungen:

1) Jeder Einzelbenutzer soll den Eindruck haben,an einer "eigenen Maschine" zu arbeiten.Dazu muß die hardware (eine oder mehrere CPU's,Hauptspeicher,Sekundärspeicher einschließlich der Magnetbänder,E/A-Geräte) auf die Benutzer aufgeteilt werden.Jeder Prozeß soll scheinbar auf einem eigenen Prozessor,auf einem "Pseudoprozessor" laufen (Definition der Begriffe Prozeß und Prozessor siehe weiter unten).Einem solchen Pseudoprozessor entspricht ein zeitlicher Anteil an einer oder mehreren CPU's.Weiterhin erhält jeder Pseudoprozessor Anteil am Hauptspeicher.Da E/A-Vorgänge parallel zu sonstigen Operationen ablaufen können und zu einer Zeit nur einen Auftrag erledigen können,der, wenn er läuft,nicht mehr unterbrochen werden kann, werden E/A-Kanälen eineindeutig Pseudoprozessoren zugeordnet,die zusätzlich zu einem Anteil am Hauptspeicher noch Hintergrundspeicher zugeordnet bekommen.

2) Gruppen von Benutzern sollen die Möglichkeit zur Kommunikation haben.Dazu ist notwendig,daß eine solche Gruppe nicht nur (nicht veränderbare) Prozedursegmente,sondern auch (veränderbare) Datensegmente gemeinsam besitzen kann.

3) Die Benutzer sollen die Möglichkeit haben,ihr "eigenes Betriebssystem" einzubringen.Solche speziellen Betriebssysteme gehen von der Annahme aus,daß ein Pseudoprozessor wie eine CPU behandelt werden kann.Die Tatsache,daß die CPU möglicherweise mehrere Pseudoprozessoren bedient,muß nicht zur Kenntnis genommen werden.Das Schreiben eines solchen Sonderbetriebssystems (welches also nicht auf der hardware,sondern auf dem Begriff des Pseudoprozessors aufbaut) wird dadurch erleichtert,daß große Teile des Betribssystems,insbesondere die Teile,welche das Benutzerprogramm an den

Pseudoprozessor adaptieren,selbst als Segmente (wie Benutzerprogramme) formuliert sind.

Bereits unter 1) sind die Begriffe Prozeß und Pseudoprozessor benutzt worden.Zunächst erscheint es naheliegend,einen Prozeß als Ablauf eines Programms auf einem Pseudoprozessor zu definieren.Es führt jedoch bereits zu Schwierigkeiten,den Begriff "Programm" exakt zu fassen.Welche Stufen von Programmen möglich sind,sei am Beispiel der TR4 erläutert:

1) Programme für den leeren Rechner;ihnen ist praktisch alles erlaubt.Auf dieser Stufe existiert noch kein Verteiler,kein System und keine Prioritäten;
2) Programme,die auf dem Verteiler aufbauen;auf dieser Stufe gibt es bereits Prioritäten,die Programme dürfen EA-Befehle,aber keine hardware-Starts für E/A-Geräte geben;
3) Programme,die auf dem System aufbauen;es sind fast alle EA-Befehle verboten,dafür stehen SYS-Befehle zur Verfügung;
4) Programme als Folge von SYS-Befehlen;die übrigen Bestandteile des Programms werden als Berechnung der Parameterversorgung für die SYS-Befehle interpretiert;
5) Programme als Abschnitt,bestehend jeweils aus den S-Steuereinheiten und der Parameterversorgung;
6) Programme,die aus Abschnitten bestehen;als Anfang uud Ende eines solchen Programms wird die Ausgabe bzw.Zurückgabe der Rechenerlaubniskarte angesehen.

Welche dieser Interpretationen gewählt wird,hängt vom Blickwinkel und den Bedürfnissen des Betrachters ab.Der designer eines compilers,ein Codierer und der Rechenzentrumsleiter werden jeweils andere Gesichtspunkte für richtig halten.

In Multics wird das Adressiersystem für die Definition der Begriffe Programm,Prozeß und Pseudoprozessor benutzt.Der Ablauf eines oder mehrerer Prozedursegmente wird als Prozeß bezeichnet,wenn für den gesamten Ablauf dieselbe Segmentnamenstafel gültig ist.Es wird die umkehrbar eindeutige Zuordnung

Prozeß ⟷ Segmentnamenstafel

Pseudoprozessor ⟷ aktive Segmentnamenstafel

getroffen.Als Programm wird nun alles angesehen,was auf einem

Pseudoprozessor laufen kann.Das wird bestimmt durch die Fähigkeiten,die ein solcher Pseudoprozessor hat.Diese sind:

1) Zugriff zu anderen Segmenten mittels verallgemeinerter Adressen.Darunter können Segmente sein,auf die auch andere Pseudoprozessoren zugreifen können.
2) Übliches Befehlsrepertoire,das die normalen arithmetischen,logischen,Shift-Befehle usw.und bis auf die in 3) und 4) angegebenen Ausnahmen keine privilegierten Befehle umfaßt.
3) Die Möglichkeit,auf Grund gewisser Befehle (z.B.Zugriff auf eine nicht im Kernspeicher stehende Seite) Sprünge auf spezielle Systemunterprogramme auszulösen.
4) Die Fähigkeit,das Betriebssystem anzurufen,um den Bereich der dem Pseudoprozessor mittels verallgemeinerter Adressen zugänglichen Segment zu verändern oder mit anderen Pseudoprozessoren oder E/A-Kanälen Signale auszutauschen.

Eine solche auf dem Begriff des Pseudoprozessors beruhende Definition von Prozeß macht Schwierigkeiten in folgenden Fällen:

1) Wenn der Prozeß selbst in mehrere "Unterprozesse" aufspaltet.In diesem Fall hat der Hauptprozeß selbst Pseudoprozessoren zu definieren,auf die Unterprozesse bezugnehmen.Diese Aufgabe auf einen Prozeß zu übertragen,kann zwar in speziellen Fällen (z.B.Test eines Betriebssystems unter der Regie eines anderen Betriebssystems) sehr vernünftig sein,verursacht im allgemeinen aber einen unverhältnismäßig großen Aufwand:z.B.ist der ganze Mechanismus zur Regievergabe,der benötigt wird, um Multitasking in PL/I im Rahmen eines Prozesses zu realisieren,bereits im Betriebssystem vorhanden und sollte daher für den Einzelprozeß nicht wiederholt werden.
2) Die Kommunikation zwischen "cooperating sequential processes" muß prinzipiell über das Betriebssystem als alleinigen Verantwortlichen für das Zusammenspiel von Prozessen erfolgen.Die direkte Kommunikation ist für solche Prozesse selbst dann nicht möglich,wenn Einzelheiten des gegenseitigen Verhaltens bei der Programmierung bekannt sind.

3) Die Kopplung Prozeß ⟷ Pseudoprozessor impliziert,daß in einem Prozeß nur ein Pseudoprozessor,also höchstens eine CPU tätig sein kann.Die Benutzung von mehreren CPU's,um parallel Unterprozesse eines Hauptprozesses durchzuführen,ist nicht möglich.

III,2 Kommunikation zwischen Prozessen

Als einführendes Beispiel für die Weitergabe von Aufträgen und Daten zwischen Prozessen werden hier die Vorgänge skizziert,die ablaufen,wenn sich ein Benutzer über seinen Fernschreiber an die Rechenanlage wendet:

Zunächst wird auf einem Pseudoprozessor der "Zuhörerprozeß" (listener) an die Regie gebracht,der vom Benutzer die zur Identifikation nötige Information beschafft.Mit dieser Information wird ein Suchprozeß gestartet,der feststellt,ob der Benutzer registriert ist,ob und wo dem Benutzer gehörige files im file-System liegen und welche Rechte er besitzt.Danach übernimmt wieder der Zuhörerprozeß die Regie,holt weitere Information und veranlaßt den "Befehlsinterpretationsprozeß" (command language interpreter) diese Information auszuwerten, was z.B. zum Start eines Übersetzers führen kann,der die nachfolgende Information als zu übersetzendes Programm verarbeitet (siehe Fig.6)

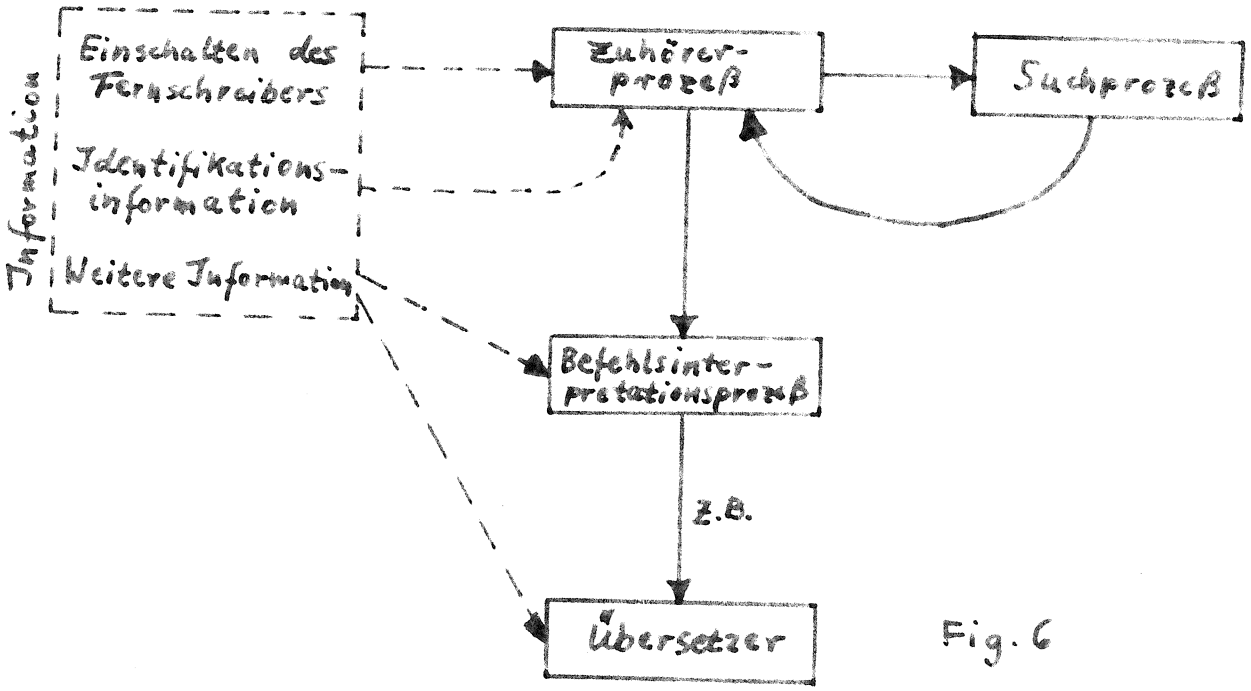

Fig. 6

Im folgenden wird die Kommunikation zwischen zwei Prozessen genauer betrachtet:

Vereinfachend wird ein Prozeß als eine Warteschlange von Aufträgen angesehen,die nacheinander abgearbeitet werden.Es wird angenommen,daß ein anderer Prozeß zu dieser Warteschlange weitere Aufträge hinzufügen kann.Es ist nun möglich,daß ein Prozeß auf die Endemeldung eines E/A-Auftrages warten muß,ehe er mit der nächsten Aufgabe seiner Auftragswarteschlange beginnen kann, oder daß der Prozeß seine Auftragswarteschlange leer findet. In solchen Fällen könnte der Prozeß solange in einer Schleife laufen,bis der E/A-Vorgang beendet bzw.bis ein neuer Auftrag eingetragen wurde.Multics verwendet eine andere Lösung dieses Problems:
Der Prozeß kann sich durch Kommando selbst blockieren (Block). Hat ein Prozeß A sich wegen Auftragsmangels selbst blockiert und setzt ein Prozeß B einen neuen Auftrag in die vorher leere Warteschlange des ProzessesA,so bemerkt B diesen Auftrag nicht und führt ihn daher auch nicht aus.Um aus dieser Situation herauszukommen,erhält B die Möglichkeit,A durch Kommando aufzuwecken (wakeup (A)).Das Aufweckkommando wirkt als Leerbefehl, falls der angesprochene Prozeß nicht blockiert war.

A sei nun ein Prozeß,der irgenwelche Ergebnisse liefert,die Prozeß B auf einem Fernschreiber ausgeben soll.Die Zusammenarbeit zwischen A und B zeigt Fig.7.B holt sich nacheinander die auszugebenden Ergebnisse,solange solche vorhanden,aus der Auftragswarteschlange,codiert um und gibt auf dem Fernschreiber aus.Ist die Warteschlange leer,so blockiert B sich selbst.Der Prozeß A weckt vorsorglich B jedesmal,wenn er Ergebnisse in die Warteschlange von B absetzt.

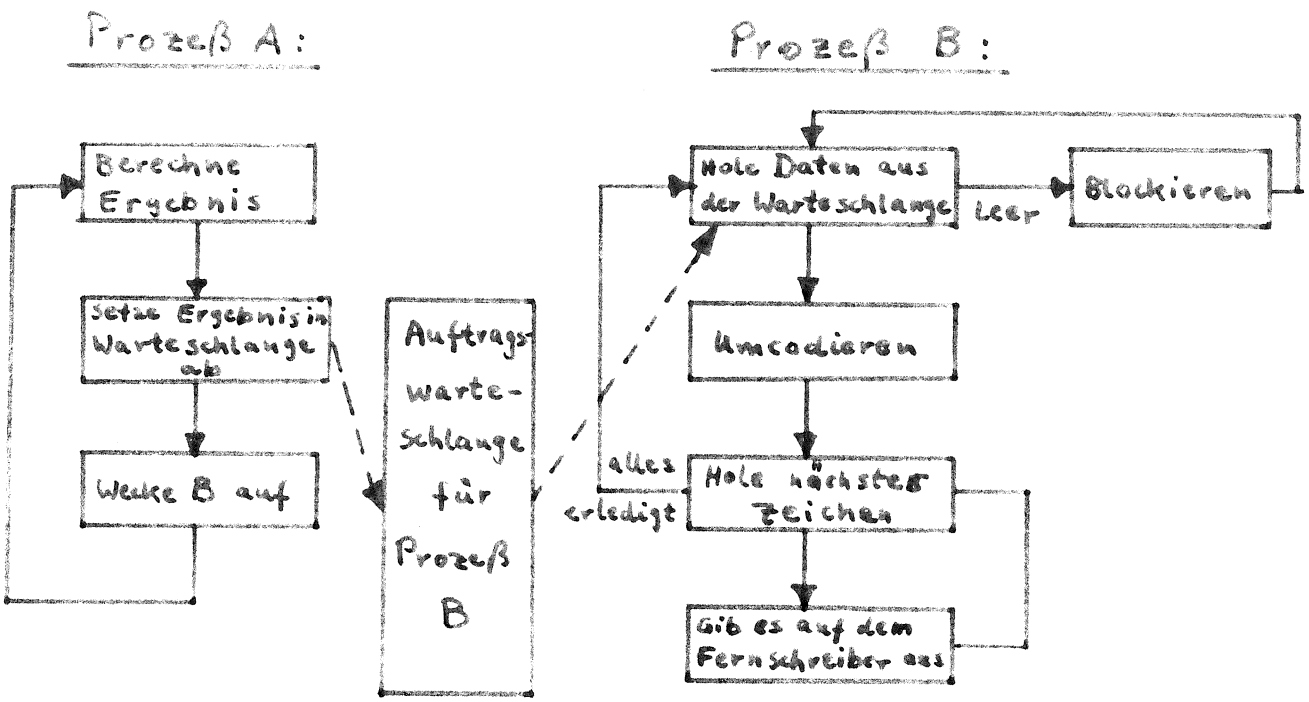

Fig. 7

Aus Speichergründen hat eine Warteschlange endliche Länge. Bis jetzt wurde stillschweigend vorausgesetzt, daß die Warteschlange für den Prozeß B nicht überläuft. Um das Überlaufen zu verhindern, soll A sich zunächst selbst blockieren, sobald es keinen Auftrag mehr in die Warteschlange absetzen kann, und warten, bis B dort wieder Platz gefunden hat. Damit A zu einem geeigneten Zeitpunkt (spätestens, wenn die Warteschlange leer ist) aufgeweckt wird, setzt es einen Hinweis, das "Weckbit" (wakeup waiting flag) in die Warteschlange von B. B weckt A auf, sobald es bei der Abarbeitung seiner Aufträge bis zu der betreffenden Stelle in seiner Warteschlange gekommen ist. Die Zusammenarbeit zwischen den Prozessen A und B unter Benutzung des Weckbits zeigt Fig. 7.

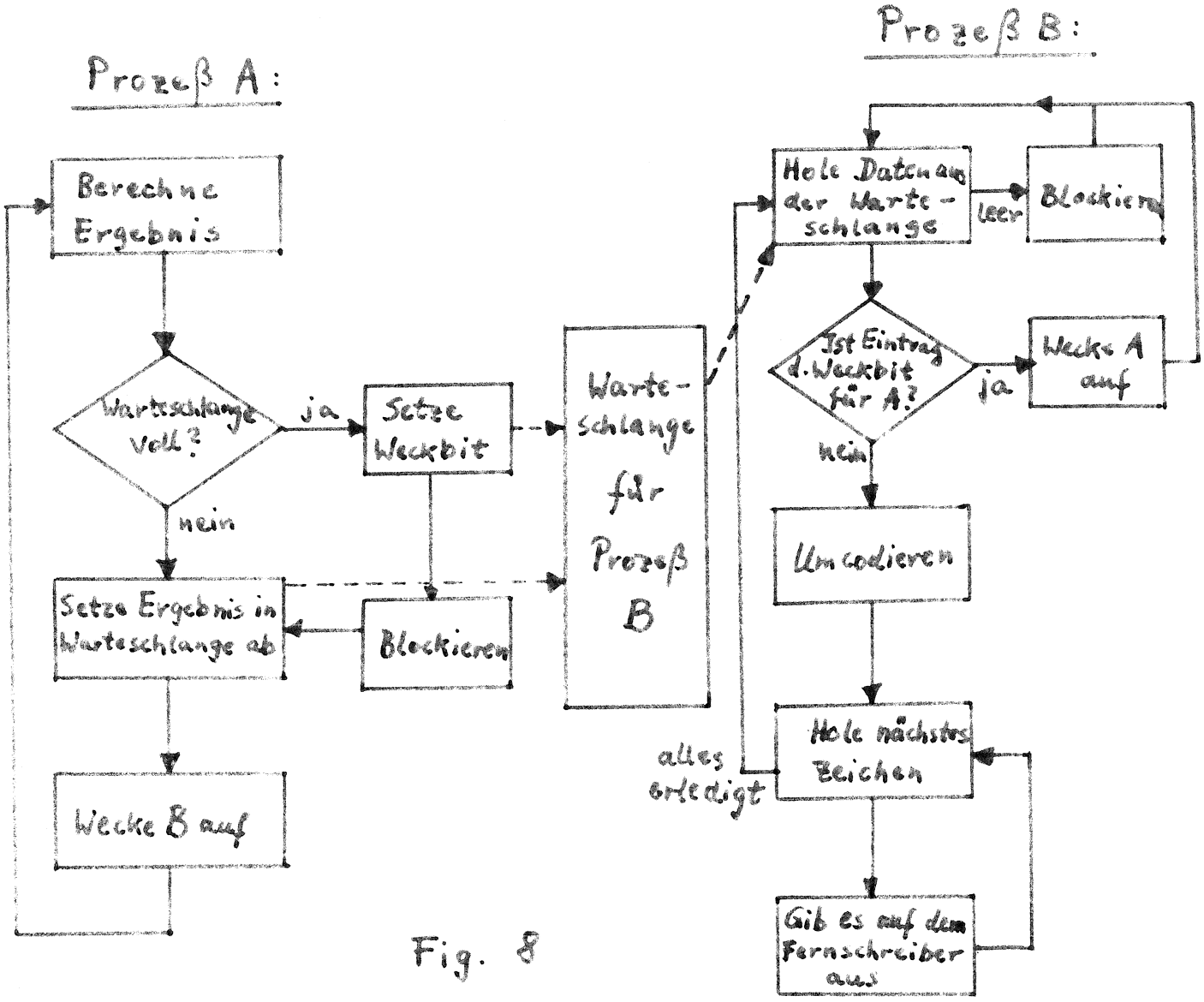

Fig. 8

Kann die Warteschlange von B nur jeweils einen Auftrag aufnehmen, so kann bei der Parallelarbeit von A und B folgende unerwünschte Situation auftreten:

B findet die Warteschlange leer und gibt Kommando, sich selbst zu blockieren. A schreibt parallel dazu einen Auftrag in die Warteschlange, gibt das Aufweckkommando für B, stellt fest, daß die Warteschlange voll ist, setzt das Weckbit und blockiert sich selbst. Beide Vorgänge können zeitlich wie folgt verschränkt sein:

1) B findet die Warteschlange leer;
2) A setzt Auftrag in die Warteschlange und weckt B (leere Anweisung, weil B noch nicht blockiert);
3) A stellt fest, daß die Warteschlange voll ist, setzt das Weckbit und blockiert sich selbst.
4) B blockiert sich selbst.

Zuletzt sind A und B blockiert (dynamic hangup).

Um diese Situation zu vermeiden, muß der Zugang zur Warteschlange gesperrt bleiben, bis B die Konsequenz aus der leeren Warteschlange gezogen und sich selbst blockiert hat. Diesen und ähnlichen Zwecken dient ein Semaphor, d.h. ein Bit, das von einem Prozeß beim Zugriff auf ein Speichergebiet auf 1 gesetzt und beim Verlassen auf 0 gelöscht wird. Versucht ein anderer Prozeß gleichfalls auf dieses Gebiet zuzugreifen, so hat er dieses Bit vorher abzufragen. Er erhält Zugang nur dann, wenn das Bit nicht gesetzt ist, andernfalls wird er in eine Warteliste eingetragen. Mit diesem Hilfsmittel kann die korrekte Synchronisation von Prozessen bewerkstelligt werden. (Genaueres über Semaphore in einem späteren Vortrag).

III,3 Aufteilung der Hardware auf die Pseudoprozessoren, insbesondere Prozessor Multiplexing

Zunächst einige kurze Bemerkungen zur E/A-Startverwaltung und Speicherverwaltung:

E/A-Startverwaltung:

Jedem E/A-Gerät wird ein Pseudoprozessor zugeordnet, der die Starts gibt. Damit ist das Problem des Starts von E/A-Vorgängen durch einen Prozeß A zurückgeführt auf die Kommunikation zwischen dem Prozeß A und einem Prozeß B, der auf dem Pseudoprozessor läuft, der dem betreffenden E/A-Gerät zugeordnet ist.

Speicherverwaltung:

Es wird angestrebt, daß von Prozessen, die rechenwillig, (Def. siehe weiter unten), aber blockiert (blocked) oder rechenfähig (ready), aber nicht an der Reihe sind, möglichst wenig Kernspeicher belegt wird. Das hat zur Folge, daß ein rechenwilliger Prozeß die in Fig. 9 angegebenen Schritte durchlaufen muß, um aktiv werden zu können. Dabei wird er immer wieder unterbrochen, wenn die angegebenen Operationen Seitentransporte erfordern.

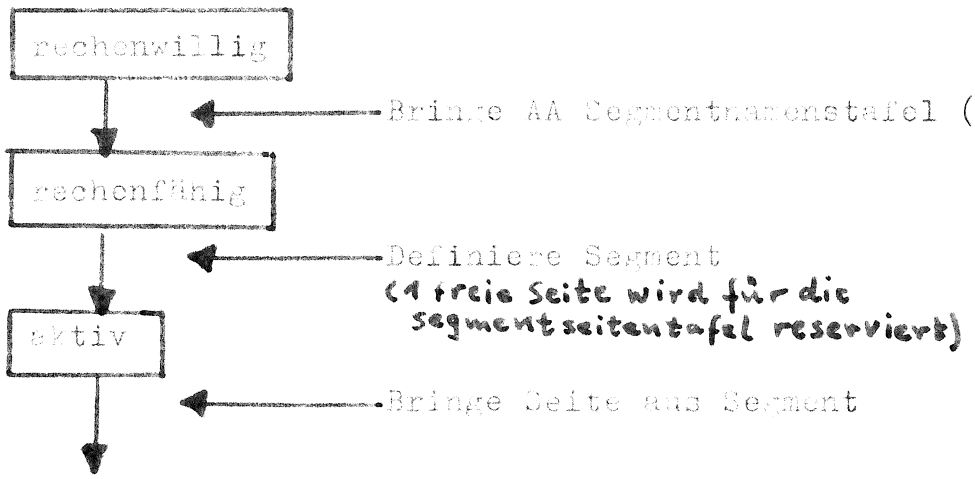

Fig.9

Um zu vermeiden, daß schließlich alle Prozesse wegen fehlender Seiten blockiert sind, versucht die Speicherverwaltung bei Bedarf zunächst Seiten von Prozessen aus dem Kernspeicher zu verdrängen, die nicht wegen fehlender Seiten blockiert sind, sondern z.B. wegen eines noch nicht beendeten E/A-Auftrages.

Prozesse, die wegen fehlender Seiten blockiert sind, heißen rechenwillig.

Die hier vorgeführte Form der Beschaffung von Seiten auf Grund von Operationen, die sonst nicht durchführbar wären, wird als dynamic loading bezeichnet.

Die Speicherverwaltung muß permanent im Kernspeicher stehen, da es sonst dazu kommen kann, daß die Speicherverwaltung unendlich oft rekursiv aufgerufen wird. (Speicherverwaltung soll fehlende Seite der Speicherverwaltung beschaffen, die ihrerseits für diesen Transport benötigt würde.)

Ebenso müssen Teile der E/A-Startverwaltung permanent im Kernspeicher stehen, besonders die Startverwaltung für den Systemhintergrundspeicher, z.B. Trommel. Sonst könnte es passieren, daß ein E/A-Auftrag für die Trommel nicht ausgeführt werden kann, weil Teile des Startprogramms auf der Trommel stehen. Um diese Teile hereinzuholen, wäre es aber notwendig, sie bereits im Kernspeicher zu haben!

Ausführlicher wird nun eingegangen auf

Unterbrechungsverarbeitung, Regie, Recke und Regierung:

Die Unterbrechungsverarbeitung verarbeitet Eingriffe (interrupts), die von peripheren Geräten, dem Recker oder einer anderen CPU verursacht wurden. Zunächst verarbeitet sie Unterbrechungen (bei der TR 440 werden diese Unterbrechungen als Alarme bezeichnet), die von der eigenen CPU zur Unterbrechung des laufenden Prozesses verursacht werden.

Regieeingriffe und Regierung regeln die Verteilung der Rechenzeit auf die Pseudoprozessoren.

Folgende Unterbrechungen können von der Unterbrechungsverarbeitung bearbeitet werden:

1) s/A-Eingriffe (externer Eingriff);
2) Weckereingriff;
3) Quit (Def. siehe unten) quittierter Eingriff;
4) Verdrängungseingriff (Def. siehe unten).

1)-3) werden als interne Eingriffe bezeichnet. Ein Prozeß A kann über das Kommando Quit (3) den Prozeß B blockieren, falls B die Regie hat. Ist B blockiert, so wird Quit als Leerbefehl. (Diese Regelung ist nur sinnvoll, wenn der Zustand "blockiert" in mehrere Unterzustände aufgeteilt und Quit einen Prozeß von einem solchen Zustand in einen anderen überführen kann, in der Multics-Literatur finden sich hierüber keine Angaben.) Quit kann benutzt werden, um einen Prozeß zu blockieren, der etwa mit falschen Daten gestartet wurde oder offensichtlich sinnlose Ergebnisse produziert. Eine Verdrängung eines laufenden Prozesses von der Regie findet dann statt, wenn ein rechenfähiger Prozeß auftaucht, den der noch zu besprechende Scheduler für "wichtiger" als alle laufenden Prozesse hält (pre-emption).

Während der Weckereingriff und der Verdrängungseingriff sich stets auf den laufenden Prozeß beziehen, können sich s/A-Eingriffe und Quit auf irgendeinen rechenwilligen, rechenfähigen oder laufenden Prozeß richten.

Alle Prozesse, die rechenwillig (oder darüberhinaus sogar rechenfähig oder laufend) sind, sind in einer Tafel der rechenwilligen Prozesse (active process table) eingetragen. Diese Tafel enthält

weiter für jeden dieser Prozesse die AA der Segmentnamenstafel (sofern im Kernspeicher), Angaben über den augenblicklichen Zustand des Prozesses und Informationen, die für die Regieübergabe an diesen Prozeß nötig sind (vgl. III,3).

Die oben angeführten Eingriffe können dazu führen, daß dem laufenden Prozeß die Regie entzogen wird. Die Regievergabe muß nun einen neuen Prozeß an die Regie bringen. Dabei werden nur die augenblicklich rechenfähigen (ready) Prozesse berücksichtigt, die in einer Liste der rechenfähigen Prozesse (ready list) eingetragen sind. Ein spezielles Unterprogramm des Systems, "Getwork", erfüllt dabei die folgenden Aufgaben:

1) aus der Liste der rechenfähigen Prozesse einen Prozeß J heraussuchen;
2) den Zustand dieses Prozesses von rechenfähig (ready) auf laufend (running) zu setzen;
3) die Übergabe der Regie an den Prozeß J zu veranlassen.

Die Regieübergabe macht Schwierigkeiten, weil sie einen Sprung auf eine Adresse verlangt, die hardwaremäßig nur mit Hilfe der Segmentnamenstafel und einer Segmentseitentafel ermittelt werden kann, wobei aber diese Tafeln nicht zum gerade laufenden Prozeß gehören und daher im Prinzip nicht zugänglich sind. Diese Schwierigkeiten könnten umgangen werden, indem die Regieübergabe in einer der drei nachstehend angegebenen Formen erfolgt:

1) Sie wird durch einen Spezialbefehl bewerkstelligt.
2) Sie wird durch ein Programm vorgenommen, das mit absoluten Adressen arbeitet und dadurch außerhalb der Prozesse steht.
3) Alle Prozesse enthalten unter der gleichen verallgemeinerten Adresse ein Regieübergabeprogramm. In diesem Fall wird die Regieübergabe einfach durch Austausch des Inhalts des Deskriptorbasisregisters ausgeführt.

In Multics wird die dritte Möglichkeit gewählt und zwar wird in der Segmentnamenstafel jedes Prozesses unter der gleichen Segmentangabe ein Segment eingetragen, das die Prozedur Swap-DBR (Laden einer neuen Adresse in das Deskriptorbasisregister, vgl. I,3) enthält. Die für die Regieübergabe nötigen absoluten Adressen

entnimmt Swap-DBR der Tafel der rechenwilligen Prozesse (active process table).

Zu untersuchen bleibt, wie das "scheduling" vorgenommen wird, d.h. wann und wo ein Prozeß in die Liste der rechenfähigen Prozesse (ready list) eingereiht wird. In Multics wird für jeden Prozeß beim Eintrag in die ready list die "Priorität", d.h. wo er in die Liste eingeordnet werden soll, neu berechnet. Jeder Prozeß soll (in gewissen Grenzen) die Möglichkeit haben, diese Operation selbst durchzuführen. Diesem Zweck dient die Konvention, daß zu jedem Prozeß A ein Prozedursegment "Schedule" gehört, das über die Segmentnamenstafel des Prozesses zugänglich ist und als erster Unterprozeß von A aufgerufen wird. Kommt ein Prozeß A neu ins System (was zum Aufruf des Systemunterprogramms "Restart" führt) oder wird ein Prozeß A rechenfähig gesetzt (was durch das Systemunterprogramm "Ready-Him" geschieht), so ruft der laufende Prozeß B das Prozedursegment Schedule des Prozesses A sozusagen als Unterprogramm auf. Dadurch wird Prozeß A in die Warteschlange (ready list) eingetragen. Anschließend setzt Prozeß B seine Tätigkeit fort, sofern er nicht von A verdrängt wird. Damit ergibt sich für den Rollenwechsel folgendes Diagramm:

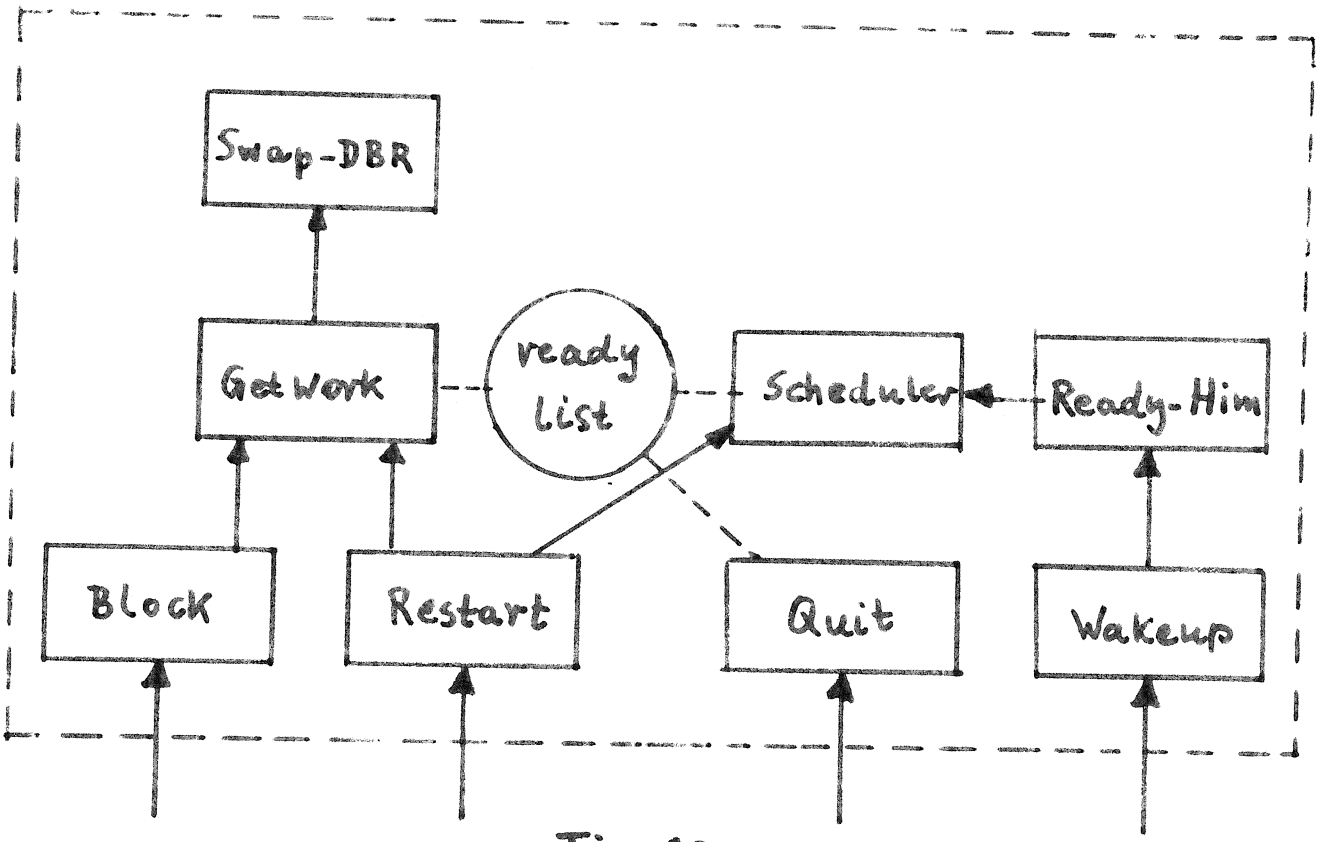

Fig. 10

In der Praxis wird nicht angenommen, daß jeder Prozeß einen privaten Scheduler-Algorithmus besitzt. Vielmehr wird man die Prozesse in Klassen einteilen, z.B. real-time-Prozesse, langlaufende Prozesse, Testprozesse; jeder Klasse ordnet man einen Scheduler-Algorithmus zu, der als gemeinsames Prozedursegment in allen Prozessen dieser Klasse benutzt wird.