RECHENZENTRUM TH MÜNCHEN
ARBEITSGRUPPE BETRIEBSSYSTEME

INTERNSCHRIFT Nr. 6

THEMA:

Grundzüge eines Dateisystems

VERFASSER:

Jammel

DATUM:

10.07.1969

FORM DER ABFASSUNG

- ENTWURF
- X AUSARBEITUNG
- ENDFORM

SACHLICHE VERBINDLICHKEIT

- ALLGEMEINE INFORMATION
  DISKUSSIONSGRUNDLAGE
- X 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

TELEFUNKEN: Datenbasis, Beschreibung für Benutzer; Stand 1.3.1969

IBM: OS 360 Data Managment.

Grundzüge eines Dateisystems

1.1 Definition

Eine Bitfolge, die (1) gemäß festen Konventionen (die i.a. Programmiersprachen entnommen sind) gebildet ist oder wird
und (2) durch spezielle hardwaremäßige Darstellung ausgezeichnet ist oder wird,
heißt ein Symbol.

Beispiele: (1) Buchstaben, etwa: S, h
Zahlen, etwa: 1, -17.3, 19_{10} -4
Namen, etwa: A13, TELEFUNKEN
Sonderzeichen, etwa: +, *, %,
Wortsymbole in speziellen Programmiersprachen, etwa: 'begin', FORMAT
Werte von Booleschen Variablen, etwa: true

(2) Inhalte von Maschinenregistern, Merklichtern, Speicherworten, Lochkartenspalten, Lochkarten-(halb)zeilen, Tetraden, Sprossen auf Lochstreifen oder Magnetband, Adress- oder Operationsteile von Befehlsworten.

1.2 Motivation

Es ist beabsichtigt, mit dem Begriff "Symbol" die ständige Berücksichtigung von elementaren Codierungen, wie z.B. Zahlen → Dezimalziffern → Binärziffern überflüssig zu machen.

./. 2

Symbolfolgen werden in Dateien temporär oder langfristig abgelegt unter zwei Gesichtspunkten:

1) Der Partitionierung (weil von einer Partitionierung einer Symbolfolge ihre Interpretation und insofern ihr Gehalt bzw. ihre Bedeutung stark abhängt.

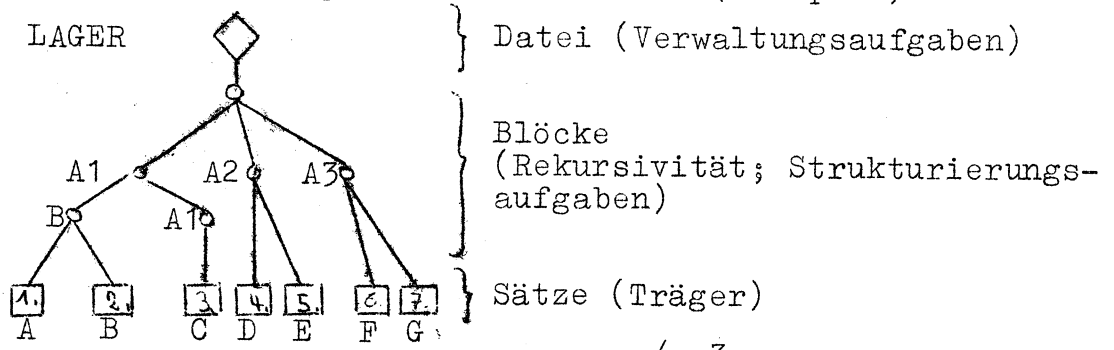
2) Der Wiederbeschaffung eigener oder fremder Symbolfolgen oder von Teilen derselben (siehe 1)).

Die Partitionierung soll nicht so geschehen, daß große Symbolfolgen durch die Datei gemäß Anweisungen der Benutzer eingeteilt werden. Vielmehr ist daran gedacht, daß der Benutzer in Dateien Symbolfolgen ablegt, die durch die Dateien keine Manipulation mehr erfahren, sondern lediglich verwaltet werden. Bei Auslieferung von n Symbolfolgen aus einer Datei soll jedoch spezifiziert werden können, ob n Symbolfolgen abgegeben werden oder ob n Symbolfolgen zu einer zusammengesetzt werden und diese dann abgegeben wird. Anstelle der ganzen Folgen sollten möglicherweise auch Teilfolgen abgerufen werden können.

2.1 Für eine Datei wird folgende syntaktische Struktur vorgeschlagen:

<Dateirumpf> → <Block>
<Block> → {<Satz>} | {<Block>}
<Satz> → {<Symbol>}

damit erhält man folgende Baumstruktur: (Beispiel)



./. 3

Eine Datei ist ein Paar: (Dateikopf, Dateirumpf)
Der Dateikopf enthält Name, Besitzer, Benutzungsanweisungen der Datei. Er kann bei Bedarf durch weitere Charakteristika ergänzt werden.
Blöcke werden über Namen erreicht, Sätze über Namen oder Nummern. Namen verweisen von einem Knotenpunkt zum andern, so daß sie nur "lokal" eindeutig sein müssen. Eine Zahl hinter einem Blocknamen verweist auf den sovielten Satz in diesem Block, wobei diesem Block evtl. nachgeordnete Blockeinteilungen ignoriert werden.

Beispiele (aus obigem Beispiel für eine Datei namens LAGER):

LAGER.A1.B	meint den Block der aus den Sätzen A und B besteht.
LAGER.A1.A1	meint den Block, der aus dem Satz C besteht
LAGER.5	meint den Satz E
LAGER.A1.2	meint den Satz B
LAGER.A1.B.B	" " " "

2.2.1 Für die Definition von Sätzen, Blöcken usw. wird folgende syntaktische Struktur vorgeschlagen:

⟨Satzdef⟩ ⇒ Satz ⟨Name⟩ := ⟨Liste⟩
⟨Blockdef⟩ ⇒ Block ⟨Name⟩ := ⟨Liste⟩
⟨Dateidef⟩ ⇒ Datei ⟨Name⟩ := ⟨Liste⟩
⟨Liste⟩ → ( ) | (⟨Ausdruck⟩ {,⟨Ausdruck⟩})
⟨Ausdruck⟩ ⇒ ⟨Name⟩ | ⟨Namensaufzählung⟩

(eine Namensaufzählung kann etwa mit Hilfe von Laufanweisungen vorgenommen werden)

Eine Liste darf nur Objekte gleichen Typs bezeichnen, also nur Namen von Symbolen bzw. von Sätzen bzw. von Blöcken bzw. von Dateien enthalten. Die Liste einer Satzdefinition enthält Namen von Symbolen und/oder Symbole oder Namen von Sätzen. Im

./. 4

letzteren Fall wird ein Satz aus den Symbolen gebildet, aus denen die in der Liste genannten Sätze bestehen. Bei einer Blockdefinition findet keine vergleichbare Entblockung statt. Die Hierarchie von Listenelementen ist die der in (2.1) definierten Hierarchie entsprechende oder eine kleinere. Bei einer kleineren wird eine vollständige Folge gebildet, wobei jeder Stufe der in der Definition spezifizierte Name zugewiesen wird.

Beispiel: Seien PI, E, GAMMA Namen von Symbolen, so hat die Definition
Block TRANS := (PI, E, GAMMA) zur Folge, daß ein Satz namens TRANS aus den Symbolen PI, E und GAMMA gebildet wird und anschließend aus diesem Satz der Block TRANS.

Die auf die obige Weise definierten Sätze und Blöcke existieren als selbständige Objekte während des Jobs, in dem sie definiert werden; ebenso Dateien, falls nicht eine längere Lebensdauer für diese spezifiziert wird.

In der Menge der Dateien und selbständigen Blöcke und Sätze müssen die Namen eindeutig sein.

Verarbeitungsbereiche werden auf folgende Weise erklärt:

⟨Puffer⟩ → Puffer ⟨Name⟩ (<⟨Zahl⟩)

Dabei wird durch ⟨Zahl⟩ die Länge des Puffers (in Ganzworten) spezifiziert. Ein Puffer verhält sich wie ein eindimensionaler Array. Dadurch können aus Pufferinhalten leicht Sätze, Blöcke usw. gebildet werden.

./. 5

2.2.2 Der Ökonomisierung der Notation (bei umständliche Block- oder Satznamen) sollen folgende Sprachelemente dienen:

⟨Satzeintrag⟩ → Satz ⟨Name⟩ := Nimm ⟨Name⟩
⟨Blockeintrag⟩ → Block ⟨Name⟩ := Nimm ⟨Name⟩ | Nimm ⟨Liste⟩

Hierbei darf die Liste lediglich Namen von Sätzen oder (aut) Blöcken enthalten.
Durch eine Nimm-Anweisung erhält das zunehmende Objekt unter dem neueingeführten Namen den Charakter eines selbständigen Objektes ("Nimm" ist in Analogie zu "take" in Euler geplant)

3.1 Für die Bearbeitung sollen folgende Möglichkeiten geschaffen werden:

Setze ⟨Name⟩ ⟨Praep⟩ ⟨Name⟩
Loesche ⟨Name⟩
Streiche ⟨Name⟩
Suche ⟨Name⟩
⟨Praep⟩ → vor | in | nach

In einer Setze-Anweisung haben die beiden Namen gleiche Hierarchie; beide bezeichnen also entweder Sätze oder Blöcke. Durch Loesche wird Information durch ?-Muster überschrieben, Streiche tilgt einen Adresseneintrag. Suche bewirkt Positionierung vor dem ersten Satz des bezeichneten Objekts, ist im Satz spezifiziert, vor diesem Satz.

Die Beziehung zwischen Dateiobjekt und Puffer wird hergestellt durch:

Schreibe $\{\{\langle Name\rangle\}_{o}^{1}\{\langle Praep\rangle\ \langle Name\rangle\}_{o}^{1}\}_{o}^{1}$
Lies $\{\{Name\}_{o}^{1}\{in\ \langle Name\rangle\}_{o}^{1}\}_{o}^{1}$

Schreiben hat die Richtung Puffer → Datei, Lesen die umgekehrte. Bei der Schreibe-Anweisung wird aus dem Pufferinhalt ein Objekt erzeugt der Hierarchie des

./. 6

spezifizierten Dateiobjekts und jenes wird geschrieben, jedoch existiert das neu erzeugte Objekt nicht als selbständiges Objekt.

Wird kein Puffer spezifiziert, so wird der dynamisch zuletzt definierte genommen, wird kein Dateiobjekt spezifiziert, so ist der auf den dynamisch letzt-erwähnten Satz folgende Satz gemeint. Betraf die dynamisch letzte Erwähnung überhaupt einen Block, so ist zu diskutieren, ob Fehlerfall vorliegen soll oder Bezug auf den letzten Satz dieses Blocks genommen werden soll. (Unter Erwähnung ist hier eine Pufferdefinition, eine Suche, Schreibe oder Lies-Anweisung gemeint)

3.2 Über Implementierungsfragen soll in einer Fortsetzungsschrift gesprochen werden.