Informatik-Spektrum 3, 31–40 (1980)

Spektrum

Das Prozeßkonzept in Betriebssystemen und Programmiersprachen II*

J. F. H. Winkler

Institut für Informatik 3 der Universität Karlsruhe

Zusammenfassung. Das Konzept des (Rechen-) Prozesses ist ein wesentliches Element von Echtzeitsprachen. Im vorliegenden Teil II der Arbeit werden die Prozeßkonzepte der sechs Echtzeitsprachen PEARL, Basis PEARL, Prozeß-Fortran, Real-Time Basic, Modula und Real-Time PL/I mit einem einheitlichen Begriffsschema beschrieben. In einer Übersicht werden die Prozeßkonzepte der sieben algorithmischen Sprachen aus Teil I und der Echtzeitsprachen anhand von 25 Merkmalen charakterisiert.

Summary. The concept of process is an essential feature of real-time languages. In the following second part of the paper the process concepts of the six real-time languages PEARL, Basic PEARL, Real-Time Fortran, Real-Time Basic, Modula, and Real-Time PL/I are described by means of a uniform framework. The process concepts of the seven algorithmic languages, which are described in part I, and the six real-time languages are characterized by 25 features. A survey of this characterization is given in a tabular summary.

Vorbemerkung

Der erste Teil dieses Aufsatzes erschien in Band 2 Heft 4 dieser Zeitschrift und enthält die Abschnitte über Betriebssysteme, Kommandosprachen und allgemeine algorithmische Programmiersprachen. Leider hat sich auf Seite 229 in (4.2-18) ein Fehler eingeschlichen. Die untere Grammatikregel sollte lauten wie folgt:

* Diese Arbeit enthält Ergebnisse aus einem mit Mitteln des Bundesministers für Forschung und Technologie (Kennzeichen DV 5.505) geförderten Forschungsvorhabens des Projektes „Prozeßlenkung mit DV-Anlagen (PDV)" im Rahmen des 3. DV-Programmes der Bundesregierung. Die Verantwortung für den Inhalt liegt jedoch allein beim Autor.

```
ausw-alternative : : =
  annahme-anw [anw-folge] |
  verzögerungs-anw [anw-folge]
```

4.3 Prozesse in Echtzeitsprachen

4.3.1 PEARL [PDV 77a]

4.3.1.1 Prozeßkonzept. Das Prozeßkonzept von PEARL läßt sich durch die folgenden drei Begriffe charakterisieren [Win 78]:

Taskdeklaration: diese hat die Form

```
bez: TASK prio; vereinb-folge anw-folge END;        (4.3-1)
```

Die Taskdeklaration ist die gemeinsame Schablone für alle dazu gehörigen Taskinkarnationen.

Taskinkarnation: die Abarbeitung einer Taskdeklaration (4.3-1) ergibt eine Taskinkarnation. Diese kann man sich als die Kombination der Routine, die durch den Taskrumpf gegeben ist, und eines Ablaufkontrollobjektes (≙ Prozeßbeschreibung in [SW 78]) vorstellen.

Taskausführung: die einmalige Ausführung des Programmes einer Taskinkarnation ist eine Taskausführung (in [PDV 77a] als „activity" bezeichnet). Die Taskausführung entspricht damit dem Prozeß, wie er hier verstanden wird.

Durch die Anweisung

```
ACTIVATE task-bez;                                  (4.3-2)
```

wird eine Taskausführung erzeugt. Ob diese Taskausführung auch sofort gestartet wird, hängt davon ab, ob eine weitere Taskausführung zu der betreffenden Taskinkarnation existiert und vor dieser Taskausfüh-

rung erzeugt wurde. Eine Taskausführung wird erst dann gestartet, wenn alle vor ihr für dieselbe Taskinkarnation erzeugten Taskausführungen beendet sind. Dieses Starten geschieht dann automatisch ohne weiteres Zutun des Programmierers. Aus der Sicht des Programmierers stellt ACTIVATE die Kombination E+S dar.

4.3.1.2 Prozeßhierarchie. Taskdeklarationen können geschachtelt sein. Außerdem können Taskdeklarationen in Prozeduren und Prozeduren in Taskdeklarationen enthalten sein. Die Unterprozeßrelation ist dynamisch definiert: Prozeß B ist unmittelbarer Unterprozeß von Prozeß A genau dann, wenn die Taskinkarnation, zu welcher B gehört, von A durch Abarbeitung der zugehörigen Taskdeklaration geschaffen wurde. Da eine Taskinkarnation ein lokales Objekt einer Blockinkarnation ist, stellt sich das Problem der impliziten Synchronisation am Blockende. Dieses ist hier so gelöst, daß eine Blockinkarnation erst dann verlassen (und damit vernichtet) werden kann, wenn zu allen darin enthaltenen Taskinkarnationen keine Taskausführungen und auch keine Einplanungen von ACTIVATE für diese Taskinkarnationen mehr existieren.

4.3.1.3 Direkte Prozeßsteuerung. Zur direkten Prozeßsteuerung dienen die folgenden Anweisungen

ACTIVATE task-bez;: s. Abschnitt 4.3.1.1

SUSPEND [task-bez];: dadurch wird ein Prozeß angehalten.

CONTINUE [task-bez];: diese Anweisung dient dem Fortsetzen eines angehaltenen Prozesses.

TERMINATE [task-bez];: dadurch wird ein Prozeß beendet und vernichtet.

Alle diese Prozeßsteueroperationen wirken automatisch auf alle Unterprozesse des angesprochenen Prozesses. Unterprozesse, die davon ausgenommen werden sollen, können in einer Ausnahmeliste (exemption) aufgeführt werden. Damit können aber nur die Unterprozesse ausgenommen werden, die von der betreffenden Programmstelle aus zugänglich sind. Bei ACTIVATE ist die Angabe einer solchen Liste nicht möglich. Sie wäre sinnlos, da die Menge der Unterprozesse eines gerade erzeugten Prozesses stets leer ist. Bei TERMINATE kann eine solche Ausnahmeliste ebenfalls nicht angegeben werden, da das Beenden und Vernichten aller Unterprozesse eines Prozesses P notwendige Voraussetzung für das Beenden und Vernichten dieses Prozesses P ist.

Wird in einer solchen Prozeßsteueranweisung kein „task-bez" angegeben, dann bezieht sich diese Anweisung auf den ausführenden Prozeß selbst. Damit ist es möglich, daß sich ein Prozeß außerhalb des Gültigkeitsbereiches der zugehörigen Taskdeklaration steuert. Dies gilt nicht für ACTIVATE; diese Operation bezieht sich auch stets auf einen anderen als den ausführenden Prozeß.

Alle oben genannten Prozeßsteueroperationen können für zukünftige Zeitpunkte eingeplant werden. Diese Einplanungen erfolgen so, daß pro Taskinkarnation für jede Prozeßsteueroperation eine endliche Menge von Zeitplänen (schedule-element) in Kraft gesetzt werden können. Ein Zeitplan stellt dabei eine endliche oder unendliche Folge von Zeitpunkten dar. Das Einrichten von Zeitplänen geschieht durch Anweisungen der Form (4.3-3).

```
schedule-element-1, . . ., schedule-element-n
proz-steuer-operat;                                    (4.3-3)
```

Durch die Anweisung

```
PREVENT task-bez;                                      (4.3-4)
```

werden alle für die betreffende Taskinkarnation gültigen Zeitpläne gestrichen. Außerdem werden alle die Taskausführungen der betreffenden Taskinkarnation vernichtet, die noch nicht gestartet wurden. Wenn eine Taskinkarnation vernichtet wird, dann werden auch alle sie betreffenden Zeitpläne gestrichen.

4.3.1.4 Indirekte Prozeßsteuerung. Zum Zwecke der Synchronisation können in PEARL Semaphore vereinbart werden und durch die Operationen REQUEST (≙ P-Operation) und RELEASE (≙ V-Operation) manipuliert werden. Außerdem gibt es Synchronisationsobjekte, die auf die Leser-Schreiberprobleme [CHP 71] zugeschnitten sind und als BOLT's bezeichnet werden. Diese Objekte können exklusiv (RESERVE) oder konkurrierend (ENTER) belegt werden, wobei die Maximalzahl der Prozesse, die gleichzeitig konkurrierend belegen können, in der Deklaration des Synchronisationsobjektes festgelegt ist. Die Freigabeoperationen sind FREE (nach RESERVE) und LEAVE (nach ENTER). Die Reihenfolge in den mit Semaphoren und Bolts verbundenen Prozeßwarteschlangen ist durch die Priorität der Prozesse bestimmt.

4.3.1.5 Priorität und Parameter. Jeder Prozeß hat eine Priorität, die dynamisch durch die CONTINUE-Anweisung geändert werden kann.

Unterschieden werden zwei Arten von Priorität: die absolute und die relative Priorität. Absolute Prioritäten sind positive ganze Zahlen, wobei größere Zahlen niedrigere Priorität bedeuten. Relative Prioritäten sind ganze Zahlen und geben die Relation der Priorität des Prozesses zu der Priorität des unmittelbaren Oberprozesses an. Negative Zahlen bedeuten höhere Priorität als die des unmittelbaren Oberprozesses und positive

Zahlen niedrigere. Prozesse, die keinen unmittelbaren Oberprozeß haben, haben stets eine absolute Priorität.

Ein Prozeß hat keine Parameter und Prozesse können nicht als Parameter von Prozeduren auftreten.

4.3.2 Basis-PEARL (DIN 66253 Teil 1)

Basis-PEARL ist eine Einschränkung von PEARL (s. Abschnitt 4.3.1). Im folgenden werden daher hauptsächlich diese Einschränkungen gegenüber PEARL aufgeführt.

4.3.2.1 Prozeßkonzept. Zu jeder in einem Basis-PEARL-Programm enthaltenen Taskdeklaration gibt es genau eine Taskinkarnation, deren Existenzdauer mit der der Programmausführung übereinstimmt. Dies ist eine Folge der Vorschrift, daß Taskdeklarationen nur im äußersten Block (auf Modulebene) stehen dürfen.

4.3.2.2 Prozeßhierarchie. Taskdeklarationen können nicht geschachtelt und auch nicht in Prozeduren enthalten sein. Daher gibt es keine Prozeßhierarchie.

4.3.2.3 Direkte Prozeßsteuerung. Da es keine Unterprozesse gibt, ist die Angabe einer Ausnahmeliste in den Prozeßsteueranweisungen nicht möglich.

Pro Taskinkarnation und pro Steueranweisung kann höchstens ein Zeitplan eingerichtet werden, d. h. in (4.3-3) ist stets $n \leq 1$.

4.3.2.4 Indirekte Prozeßsteuerung. Zur Synchronisation sind nur die Semaphore vorhanden.

4.3.2.5 Priorität und Parameter. Es gibt nur die absolute Priorität, und zwar in statischer Form derart, daß alle Taskausführungen einer Taskinkarnation dieselbe Priorität haben, die nur durch eine Modifikation des Programmtextes geändert werden kann.

4.3.3 Prozeß-Fortran

Zu einer Echtzeitsprache kann man auch gelangen, indem man eine existierende Programmiersprache um entsprechende Sprachelemente erweitert. Dieser Weg wurde insbesondere bei Fortran, das immer noch eine weite Verbreitung hat, mehrfach beschritten [VDI 78: 3,34; CCH 78: 24; PSI 78: 4. . 6]. Im folgenden wird Prozeß-FORTRAN 75 [VDI 78] beschrieben, das derzeit von einer Reihe von Prozeßrechnerherstellern angeboten wird [AEG 76: 80; DT 76]. Eine andere Erweiterung von Fortran für die Echtzeitprogrammierung wird im Rahmen des International Purdue Workshop on Industrial Computer Systems unter Beteiligung der ISA (Instrument Society of America) definiert [ABC 79; ISA 78]. Diese beiden Fortran-Erweiterungen unterscheiden sich von [VDI 78] hauptsächlich im Vorhandensein von Mechanismen zur indirekten Prozeßsteuerung (Semaphor in [ABC 79], resourcemark in [ISA 78]).

4.3.3.1 Prozeßkonzept. Als zentraler Begriff des nebenläufigen Ablaufgeschehens wird in Prozeß-Fortran der Begriff „TASK" eingeführt und wie folgt definiert:

„Eine TASK ist ein dem Betriebssystem bekannter Auftrag, dem eine bestimmte Befehlsfolge und eine bestimmte Priorität zugeordnet sind [VDI 78: 5]."

Dieser Begriff deckt sich nicht mit dem hier verwendeten Prozeßbegriff, der auf die einmalige Ausführung eines Programmes (einer Prozedur) abhebt, denn eine TASK kann mehrfach gestartet werden. Der hier verwendete Begriff „TASK" entspricht daher eher dem in 4.3.1.1 eingeführten Begriff der Taskinkarnation. In Prozeß-Fortran wird nicht definiert, wie eine TASK gebildet wird, sondern bei Beginn der Programmausführung wird eine feste Menge von TASKs als existent vorausgesetzt, die betriebssystemspezifisch mittels Zahlen oder Zeichenketten identifiziert werden. Das Erzeugen eines Prozesses erfolgt durch einen der folgenden Prozeduraufrufe: CALL START, CALL TRNON, CALL CYCLE, CALL TCYCLE und CALL CON, wobei jeweils verschiedene Parameter anzugeben sind.

4.3.3.2 Prozeßhierarchie. Eine Prozeßhierarchie existiert nicht. Damit soll erreicht werden, daß Prozeß-FORTRAN 75 zusammen mit unterschiedlichen Echtzeitbetriebssystemen verwendet werden kann.

4.3.3.3 Direkte Prozeßsteuerung. Die direkte Prozeßsteuerung erfolgt durch die folgenden Prozeduraufrufe:

CALL START:	erzeugen eines Prozesses und starten nach einer angegebenen Zeitspanne.
CALL TRNON:	erzeugen eines Prozesses und starten zu einer angegebenen Tageszeit.
CALL WAIT:	damit kann sich ein Prozeß um eine angegebene Zeitspanne verzögern.
CALL HOLD:	anhalten des ausführenden Prozesses.
CALL RELSE:	fortsetzen eines Prozesses, der infolge eines CALL HOLD angehalten ist.
CALL CYCLE:	erzeugen und starten eines Prozesses und einplanen weiterer sol-

cher Starts für unendlich viele äquidistante Zeitpunkte in der Zukunft.

CALL TCYCLE: erzeugen eines Prozesses und starten zu einem angegebenen Zeitpunkt, außerdem einplanen weiterer solcher Starts für unendlich viele äquidistante Zeitpunkte in der Zukunft.

CALL CANCL: streichen von Einplanungen, die durch START, TRNON, CYCLE oder TCYCLE erfolgt sind.

CALL CON: einplanen, daß bei Auftreten eines angegebenen äußeren Ereignisses ein Prozeß zu erzeugen und zu starten ist. Dies wird dann bei jedem Auftreten dieses Ereignisses durchgeführt.

CALL UNCON: streichen der durch CALL CON erfolgten Einplanung.

Durch die Fortran-Anweisung „STOP" kann sich ein Prozeß selbst beenden.

4.3.3.4 Indirekte Prozeßsteuerung. Eine Synchronisation ist nur durch exklusives Belegen von Dateien möglich. Da der Nichterfolg durch den Wert eines Ergebnisparameters mitgeteilt wird, d. h. nicht in einen Wartezustand führt, ist diese Art der Synchronisation mit aktivem Warten verbunden.

4.3.3.5 Priorität und Parameter. Den TASK's, d. h. Prozeßklassen, ist eine positive ganze Zahl als Priorität zugeordnet, wobei größere Zahlen niedrigere Priorität bedeuten. Die Priorität ist allerdings im Prozeß Fortran-Programm nicht zugänglich.

Prozesse haben keine Parameter und können auch nicht als Prozedurparameter auftreten.

4.3.4 Real-Time Basic

Ebenso wie für Fortran gibt es auch für Basic eine Reihe von Erweiterungen für die Echtzeitprogrammierung [Haa 76; HS 77:148..156; PSI 78:4..6]. Es entstanden dabei eine Vielzahl unterschiedlicher Versionen. Im folgenden wird der Normvorschlag [Ind 79, 79a] behandelt. Der erste Teil dieses Vorschlages [Ind 79] ist als ANSI draft standard angenommen [LT 79].

4.3.4.1 Prozeßkonzept. Ein Programm besteht aus einem Hauptprogramm und einer (evt. leeren) Folge von Parallelabschnitten („parallel section" in [Ind 79]) der Form (4.3-5).

```
zeil-nr PARACT abschn-bez [URGENCY num-ausdr]
        .
        .                                        (4.3-5)
        .
zeil-nr PAREND
```

Ein Programmsystem besteht aus einer Menge von (separat übersetzten) Programmen.

Das Hauptprogramm eines Programmes wird während einer Programmausführung genau einmal ausgeführt und kann daher mit einem Prozeß im Sinne von Abschnitt 1 identifiziert werden. Ein Parallelabschnitt kann während einer Programmausführung beliebig oft ausgeführt werden und entspricht daher einer Taskinkarnation im Sinne von Abschnitt 4.3.1.1.

Ein Prozeß entsteht also entweder durch das Starten eines Hauptprogrammes oder durch das Starten eines Parallelabschnittes. Zu einem Zeitpunkt kann es zu einem Hauptprogramm bzw. Parallelabschnitt höchstens einen Prozeß geben. Der zu einem Hauptprogramm gehörende Prozeß ist anonym, während die anderen Prozesse jeweils durch die Abschnittsnamen bezeichnet werden. Wie die verschiedenen Hauptprogramme eines Programmsystems benannt und gestartet werden, wird in einer zur Zeit entstehenden Überarbeitung von [Ind 79a] festgelegt werden.

4.3.4.2 Prozeßhierarchie. Eine Prozeßhierarchie existiert nicht.

4.3.4.3 Direkte Prozeßsteuerung. Zur direkten Prozeßsteuerung dienen die folgenden Anweisungen:

START: dadurch wird ein neuer Prozeß erzeugt und gestartet, der sowohl zu einem Hauptprogramm als auch zu einem Parallelabschnitt gehören kann.

WAIT: dadurch kann sich ein Prozeß um ein bestimmtes Zeitintervall oder bis zu einem bestimmten Zeitpunkt verzögern.

PARSTOP: durch die Ausführung dieser Anweisung wird der sie initiierende Prozeß beendet und vernichtet.

STOP: dadurch werden alle Prozesse eines Programmes beendet und vernichtet.

Die START-Anweisung kann für einen Zeitpunkt in der Zukunft eingeplant werden, wobei dieser Zeitpunkt durch ein Zeitintervall, eine absolute Zeitangabe oder ein Ereignis (event) bestimmt werden kann. Solche Einplanungen können durch die CANCEL-Anweisung wieder gestrichen werden.

4.3.4.4 Indirekte Prozeßsteuerung. Bei der indirekten Prozeßsteuerung sind je ein Mechanismus zur Synchronisation und zur Kommunikation vorgesehen.

Die Synchronisation erfolgt mit Hilfe von Ereignisobjekten (event), die entweder vordefinierte Unterbrechungseingänge oder im Programm vereinbarte binäre Semaphore sind. Ein Unterbrechungseingang kann per Programm auf „offen" (CONNECT) oder „geschlossen" (DISCONNECT) gesetzt werden.

Das Warten auf ein Ereignis erfolgt mit der WAIT-Anweisung in der Form (4.3-6).

```
WAIT EVENT ereign-bez
[TIMEOUT zeit-ausdr]                    (4.3-6)
```

Wenn das Ereignis nicht während des nach TIMEOUT angegebenen Zeitintervalles eintritt, wird eine Ausnahmebedingung gesetzt.

Ein Ereignis tritt auf, wenn die Anweisung (4.3-7) ausgeführt wird, oder wenn die entsprechende Unterbrechung auftritt.

```
SIGNAL ereign-bez                       (4.3-7)
```

In der SIGNAL-Anweisung darf auch ein Unterbrechungseingang genannt werden. Pro Ereignis wird eine WAIT-Anweisung befriedigt. Das Entstauen der wartenden Prozesse erfolgt nichtdeterministisch.

Die Kommunikation erfolgt mit den Anweisungen (4.3-8) und (4.3-9) über vorher definierte Nachrichtenkanäle.

```
SEND TO kanal-bez FROM ausdr-liste      (4.3-8)
RECEIVE FROM kanal-bez TO var-liste     (4.3-9)
```

Ein Nachrichtenkanal bildet einen Stauraum für Nachrichten und Empfangswünsche. Ein Prozeß, der einen Empfangswunsch an einen leeren Kanal richtet, wird so lange angehalten, bis eine Nachricht an diesen Kanal gesendet wird. Ein sendender Prozeß wird nie angehalten, so daß in der Regel kein Rendezvous der kommunizierenden Prozesse stattfindet. Das Entstauen gestauter Nachrichten erfolgt nichtdeterministisch.

Kommunikation ist auch möglich mittels Ereignisobjekten der Art Semaphor (in [Ind 79] als „event" bezeichnet) und globalen Datenobjekten.

4.3.4.5 Priorität und Parameter. Einem Parallelabschnitt kann nach (4.3-5) eine Priorität zugeordnet sein, die bei jeder START-Anweisung neu berechnet wird. Für einen einzelnen Prozeß ist die Priorität daher stets fest.

Prozesse haben keine Parameter und können auch nicht als Parameter von Prozeduren auftreten.

4.3.5 Modula

Von Modula existieren zwei verschiedene Versionen: Modula [Wir 77] und Modula-2 [Wir 78]. Im folgenden wird Modula behandelt. Der wesentliche Unterschied zwischen beiden Sprachen liegt im Bereich der Prozeßsteuerung. Modula-2 ist auf Monoprozessor-Rechenanlagen zugeschnitten und enthält nur ein (symmetrisches) Koroutinenkonzept, aber kein Prozeßkonzept im allgemeinen Sinne.

4.3.5.1 Prozeßkonzept. Ein Prozeßtyp wird durch eine Prozeßdeklaration definiert, die im wesentlichen die Form (4.3-10) hat.

```
PROCESS prozedur-deklaration            (4.3-10)
```

Durch einen Prozeßaufruf, der dieselbe Form wie ein Prozeduraufruf hat, wird zu einem Prozeßtyp ein (anonymer) Prozeß erzeugt und gestartet. Es können gleichzeitig mehrere Prozesse eines Prozeßtyps existieren. Eine Anweisung zum Vernichten von Prozessen ist nicht vorgesehen.

4.3.5.2 Prozeßhierarchie. Prozeßdeklarationen können nur im äußersten Block eines Programmes vorkommen und können daher auch nicht geschachtelt sein. Es existiert daher keine Prozeßhierarchie. Da auch Prozeßaufrufe nur im äußersten Block stehen dürfen, kann auch über die Aufruffolge keine Hierarchie definiert werden.

4.3.5.3 Direkte Prozeßsteuerung. Außer dem Prozeßaufruf, der das Erzeugen und Starten eines Prozesses bewirkt, gibt es keine Mittel zur direkten Prozeßsteuerung. Wenn ein Prozeß das Ende des Programmes seines Prozeßtyps erreicht, wird er beendet aber nicht vernichtet.

4.3.5.4 Indirekte Prozeßsteuerung. Zur Synchronisation der Prozesse sind sogenannte „interface-module" vorhanden, die im wesentlichen den Monitoren von Hoare und Brinch Hansen [Hoa 74; Bri 75c] entsprechen. Der wesentliche Unterschied ist der, daß bei der wait-Operation eine Warteschlangenpriorität in Form einer positiven, ganzen Zahl angegeben werden kann. Bei einer send-Operation für ein Signal wird der Prozeß mit der kleinsten Prioritätszahl unter den auf dieses Signal wartenden Prozessen fortgesetzt. Wenn mindestens ein wartender Prozeß existiert, dann wird der sendende Prozeß angehalten.

4.3.5.5 Priorität und Parameter. Prozesse haben im allgemeinen keine Priorität. Lediglich ein sogenannter „device-process", der in einer PDP 11-spezifischen Spracherweiterung enthalten sein kann, kann eine Priorität haben.

Prozesse können Datenparameter haben, wobei sowohl Wert- als auch Referenzübergabe auftreten dürfen. Prozesse können nicht als Parameter auftreten.

4.3.6 Real-Time PL/I [Bar 79]

In [Bar 79] wird eine Erweiterung von Standard-PL/I (ANS X3.53-1976) für Zwecke der Echtzeitprogrammierung vorgeschlagen. Standard PL/I und der damit identische Standard ECMA-50 vom 16. Dez. 1976 enthalten die in 4.2.1 beschriebenen Sprachelemente zur Formulierung und Steuerung paralleler Prozesse nicht.

4.3.6.1 Prozeßkonzept. Ein „external entry point" eines PL/I-Programmes kann mit dem PROGRAM-Attribut versehen werden. Solcherart gekennzeichnete Prozeduren können als eigenständige Prozesse ausgeführt werden. Ein Prozeß (in [Bar 79] als „task" bezeichnet) ist definiert als die einmalige Ausführung einer solchen Prozedur. Dieser Prozeßbegriff deckt sich daher mit der Definition in Abschnitt 1.

Durch die SCHEDULE-Anweisung (4.3-11) wird ein Prozeß erzeugt, gestartet und eventuell sogleich angehalten.

```
SCHEDULE prog-bez [PRIORITY(n)]
[AS(pn)] [WHEN (ev-liste)]                    (4.3-11)
[STATUS(s)] [OPTIONS(impl-defined)]
```

Jede andere Reihenfolge der optionalen Elemente in (4.3-11) ist ebenfalls zulässig. „prog-bez" charakterisiert das Programm des Prozesses. Dabei muß es sich um einen mit dem PROGAM-Attribut versehenen „external entry point" handeln. Ein Prozeß kann durch eine ganze Zahl eindeutig bezeichnet werden. Wenn dies der Fall sein soll, dann muß der erzeugende Prozeß diese Zahl als „pn" in der AS-Klausel angeben. Zu jedem Zeitpunkt müssen die pn-Werte aller so bezeichneten und noch existierenden Prozesse voneinander verschieden sein. Wenn die WHEN-Klausel angegeben ist, dann wird der neu erzeugte Prozeß so lange angehalten, bis mindestens eine der Bedingungen in der „ev-liste" wahr ist. Solche Bedingungen können sein:

a) Relation zwischen einem gemeinsam benutzten Datenobjekt (s. Abschnitt 4.3.6.4) und dem Wert eines Ausdrucks,

b) Erreichen eines bestimmten Zeitpunktes,

c) Auftreten einer Unterbrechung oder einer Dateneingabe,

d) Beendigung eines Prozesses.

Durch die STATUS-Klausel kann einem Prozeß eine gemeinsam benutzte Ganzzahlvariable „s" (Statusvariable) zugeordnet werden, die bei der Beendigung des Prozesses auf einen definierten Wert gesetzt wird (s. Abschnitt 4.3.6.3).

4.3.6.2 Prozeßhierarchie. Eine Prozeßhierarchie existiert weder im statischen noch im dynamischen Sinne. Da externe Prozeduren nicht in Blöcken enthalten sein können, existiert das Problem der impliziten Synchronisation am Blockende hier nicht.

4.3.6.3 Direkte Prozeßsteuerung. Ein Prozeß kann sich beenden und vernichten durch Ausführung der STOP-Anweisung oder durch Verlassen seiner (dynamisch) ersten Prozedur. Diese Arten der Beendigung werden als normale Beendigungen bezeichnet. Ist dem Prozeß eine Statusvariable zugeordnet, dann wird diese auf den Wert 1 gesetzt, wenn sie sich im Zustand „abandoned" (s. Abschnitt 4.3.6.4) befindet oder den Wert 0 hat. Da die Statusvariable dazu exklusiv belegt (s. Abschnitt 4.3.6.4) werden muß, und eine Statusvariable von anderen Prozessen als normales, gemeinsam benutztes Objekt verwendet werden kann, kann sich dieses Setzen der Statusvariablen beliebig lange verzögern.

Durch die ABORT-Anweisung (4.3-12) können Prozesse abgebrochen und vernichtet werden.

```
ABORT [(integer-ausdr)];                      (4.3-12)
```

Wenn der „integer-ausdr" fehlt, dann bezieht sich die Anweisung auf den Prozeß, der sie ausführt. Diese Art der Beendigung wird als abnormale Beendigung bezeichnet. Ist einem abnormal beendeten Prozeß eine Statusvariable zugeordnet, dann wird dieser so bald wie möglich ein (implementationsabhängiger) negativer Wert zugewiesen.

Durch die WAIT-Anweisung (4.3-13) kann sich ein Prozeß verzögern, bis mindestens eine der Bedingungen in der „ev-liste" wahr ist.

```
WAIT (ev-liste) [EVENTNO (n)];                (4.3-13)
```

Mit der EVENTNO-Klausel kann festgestellt werden, welche der Bedingungen in „ev-liste" wahr ist. Deren Nummer wird der Variablen „n" zugewiesen. Die Bedingungen in „ev-liste" werden dabei, als mit 1 beginnend durchnumeriert gedacht. Wenn mehrere Bedingungen wahr sind, dann wird auf implementationsabhängige Art und Weise die Nummer einer dieser Bedingungen der Variablen „n" zugewiesen.

4.3.6.4 Indirekte Prozeßsteuerung. Die Kommunikation zwischen Prozessen erfolgt über gemeinsam benutzte Datenobjekte (SHARED-Attribut). Dabei kann es sich um einfache oder strukturierte Objekte vom Typ „computational" oder um Dateikonstanten handeln. Wenn eine Dateikonstante das SHAREDRECORDS-Attribut besitzt, dann können einzelne Sätze dieser Datei zur Kommunikation benutzt werden.

Der kontrollierte Zugriff zu den gemeinsam benutzten Objekten, der für eine ordnungsgemäße Kommunikation notwendig ist, ist innerhalb der LOCK-Anweisung (4.3-14) möglich.

```
LOCK {(schr-liste) [READONLY (les-liste)] |
READONLY (les-liste)} [TIMEOUT (i[,s])]          (4.3-14)
THEN anweisung
[ELSE balancierte-anweisung]
```

Wenn die TIMEOUT-Klausel vorhanden ist, dann muß auch die ELSE-Klausel vorhanden sein.

Die in der „schr-liste" genannten Objekte werden für exklusiven Zugriff belegt, die in der „les-liste" genannten für konkurrierenden Zugriff. Wenn der zwischen LOCK und THEN spezifizierte Zugriff möglich ist, werden die entsprechenden Belegungen vorgenommen, die THEN-Klausel ausgeführt, die Belegungen wieder aufgehoben und mit der Folgeanweisung fortgefahren. Innerhalb der THEN-Klausel ist ein beliebiger Zugriff auf die belegten Objekte möglich; der Programmierer ist jedoch dafür verantwortlich, daß die Benutzung mit der Belegungsart verträglich ist.

Wenn der gewünschte Zugriff nicht möglich ist und eine TIMEOUT-Klausel vorhanden ist, dann wird versucht, in dem in der TIMEOUT-Klausel spezifizierten Zeitintervall (i bzw. i*10^s Sekunden) den Zugriff zu realisieren. Ist dies nicht möglich, dann wird nach Ablauf dieses Zeitintervalles die ELSE-Klausel ausgeführt und mit der Folgeanweisung fortgefahren. Ist die TIMEOUT-Klausel nicht angegeben, dann ist das oben erwähnte Zeitintervall 0 Zeiteinheiten lang. Wenn keine ELSE-Klausel vorhanden ist, dann wird der Prozeß so lange angehalten, bis der gewünschte Zugriff möglich ist. Wenn zu einem Zeitpunkt sich gegenseitig ausschließende Belegungen mehrererLOCK-Anweisungen (die dann zu verschiedenen Prozessen gehören) erfüllbar sind, dann ist nicht definiert, welche dieser Belegungen vorgenommen wird.

Bei normaler Beendigung eines Prozesses innerhalb eines kritischen Abschnittes (THEN-Teil von (4.3-14)) werden alle Belegungen aufgehoben. Bei abnormaler Beendigung werden die konkurrierend belegten Objekte freigegeben und die exklusiv belegten Objekte in den Zustand „abandoned" versetzt. Dieser Zustand kann mittels der ABANDONED-Funktion innerhalb eines kritischen Abschnittes, in welchem das betreffende Objekt exklusiv belegt ist, getestet werden. Der Zustand „abandoned" trägt dem Umstand Rechnung, daß bei einer abnormalen Beendigung (die im allgemeinen asynchron erfolgt) ein exklusiv belegtes Objekt sich in einem inkonsistenten Zustand befinden kann.

4.3.6.5 Priorität und Parameter. Als Priorität werden positive, ganze Zahlen aus dem Intervall [1, n] verwendet, wobei n implementationsabhängig ist. Die Priorität kann bei der Erzeugung eines Prozesses (4.3-11) angegeben werden und anschließend beliebig gelesen und gesetzt werden. Sie hat nur Bedeutung für die Vergabe der CPU.

Prozesse können nur Wertparameter von arithmetischer Art oder der Art „string" haben. Prozesse können nicht als Parameter auftreten.

4.4 Übersicht

In Tabelle 4.1 sind die in den Abschnitten 4.2 und 4.3 behandelten Prozeßkonzepte anhand von 25 Kriterien charakterisiert.

Die Kriterien 1 bis 7 charakterisieren das Prozeßkonzept, eingeschlossen die Bereiche Hierarchie, Priorität und Parameter.

Die Kriterien 8 bis 20 betreffen die Anweisungen zur direkten Prozeßsteuerung. Dabei entsprechen 8 bis 13 den sechs in Abschnitt 2.3 angegebenen Operationen. Die Kriterien 14 (E+S) und 15 (B+V) gelten für den Fall, daß die betreffenden Operationen nur in diesen Kombinationen und damit nicht einzeln verfügbar sind (s. auch Abschnitt 4.2.1.3). Die Kriterien 17 bis 20 betreffen die zeitlichen Einplanungen von Operationen zur direkten Prozeßsteuerung.

Die dritte Gruppe von Kriterien (21 bis 25) betreffen die indirekte Prozeßsteuerung durch Synchronisation und Kommunikation.

5. Zusammenfassung

Die Übersicht in Abschnitt 4.4 zeigt, daß die verschiedenen Prozeßkonzepte stark variieren. Dies gilt für alle Teilbereiche wie direkte Prozeßsteuerung, Synchronisation und zeitbezogene Konzepte.

PEARL und Real-Time Basic weisen viele Funktionen im Bereich der Prozeßsteuerung auf, während Modula am sparsamsten ausgestattet ist und z. B. das Konzept der Zeit überhaupt nicht enthält. Keines der betrachteten Prozeßkonzepte stellt in allen Teilbereichen befriedigende Fähigkeiten zur Verfügung. Während PEARL im Bereich der direkten Prozeßsteuerung und ihrer Einplanung viele Hilfsmittel bereitstellt, bietet es auf dem Gebiet von Synchronisation und Kommunikation nur relativ einfache Mittel. Auf diesem Gebiet weisen Ada, Platon, Real-Time Basic und Real-Time PL/I interessante Lösungen auf. Ada enthält ein elegantes Konzept für die Kommunikation auf höherem Niveau in Form des einseitig anonymen Rendezvous mit disjunktivem Empfangen. In Real-Time PL/I sind zur Sychronisation kritische Abschnitte in Form der LOCK-Anweisung realisiert, wobei auch mehr pragmatische Aspekte („abandoned"-Zustand) berücksichtigt werden, die bei vielen rein theoretischen Untersuchungen zur Synchronisation paralleler Prozesse nicht betrachtet werden. In Platon wird die Kommunikation zwischen Prozessen durch eine Variante der Zuweisung realisiert, bei welcher die rechte Seite *nicht* kopiert wird,

Tabelle 4.1. Charakterisierung der Prozeßkonzepte

Nr	Konzept	PL/1	Burr. Algol	Algol 68	Conc. Pascal	Platon	OSL/2	Ada	PEARL	Basis PEARL	Prozeß Fortran 75	Real-Time Basic	Modula	Real-Time PL/1
1	Programm und PB getrennt ?	J	J	N	N	N	J	N	N	N	N	N	N	J
2	Anzahl der PB statisch fest ?	N	N	Ø	J	J	N	N	N	J	J	J	N	N
3	Anzahl der Prozesse statisch fest ?	N	N	J	J	N	N	N	N	N	N	N	N	N
4	Hierarchie: stat dyn	J J	J J	J J	J N	J N	N J	J J	J J	N N	N N	N N	N N	N N
5	Priorität: stat dyn	N J	J N	N N	N N	N N	N N	N J	N J	J N	N N	J N	N N	N J
6	Param. für Prozeß	J	J	J	J	J	J	N	N	N	N	N	J	J
7	Prozeß als Param.	J	J	N	N	N	J	N	N	N	N	N	N	N
8	Erzeugen	N	N	N	F	F	N	N	FE	FE	N	N	N	N
9	Vernichten	N	N	N	N	N	N	N	FE	F	N	N	N	N
10	Starten	N	N	N	F	J	N	N	I	I	N	N	N	N
11	Beenden	N	N	N	S	S	N	N	N	N	N	N	N	N
12	Anhalten	N	J	N	N	J	J	N	E	S	S	N	N	N
13	Fortsetzen	N	J	N	N	J	J	N	E	E	F	N	N	N
14	E + S	F	F	F	N	N	F	F	N	N	FE	FE	F	FE
15	B + V	J	J	J	N	J	J	J	E	J	J	S	N	J
16	Verzögern	S	S	N	N	N	N	J	J	S	S	S	N	S
17	Einplanen	N	N	N	N	N	N	N	J	J	J	J	N	J
18	Ausplanen gezielt global	N N	N N	N N	N N	N N	N N	N N	N J	N J	J N	J N	N N	N N
19	Uhrzeitfunktion	J	J	N	N	N	N	J	N	N	J	J	N	J
20	Datumfunktion	J	J	N	N	N	N	N	N	N	J	J	N	J
21	Semaphor	N	J	J	N	J	J	J	J	J	N	J	N	N
22	Bolt	N	N	N	N	N	N	N	J	N	N	N	N	N
23	Monitor	N	N	N	J	N	N	N	N	N	N	N	J	N
24	Krit. Abschnitt	N	N	N	N	N	N	N	N	N	N	N	N	J
25	Prozeßkommunikation	N	N	N	N	J	N	J	N	N	N	J	N	N

Die in Tabelle 4.1 verwendeten Abkürzungen haben folgende Bedeutung:

0: Kriterium nicht anwendbar
J: Ja
N: Nein
E: Operation ist vorhanden und kann auch eingeplant werden
F: Operation ist vorhanden und kann nur fremdbezogen, d. h. auf andere Prozesse ausgeübt werden [Win 79:42]
S: Operation ist vorhanden und kann nur selbstbezogen, d. h. auf den ausführenden Prozeß ausgeübt werden [Win 79:42]
I: Operation wird implizit von der Maschine M angewendet

sondern eine echte Übergabe stattfindet (s. SIGNAL-Anweisung). Real-Time Basic enthält sowohl Funktionen zur reinen Synchronisation als auch zur Kommunikation.

Obwohl PEARL auf dem Gebiet der Zeitfunktionen die meisten Funktionen bietet, lassen sich leicht Situationen angeben, in welchen diese Hilfsmittel nicht ausreichen [Eic 75; Lau 78].

Für eine kritische Durchsicht des Manuskriptes und wertvolle Hinweise bin ich C. Stoffel und den Gutachtern zu Dank verpflichtet. Durch Überlassung von Unterlagen über einzelne Sprachen und Hinweise zu den entsprechenden Abschnitten haben mich S. Eichentopf (PEARL), V. H. Haase (Real-Time Basic) und G. Heller (Real-Time Fortran) unterstützt. Ihnen gebührt ebenso mein Dank.

Literatur

ABC 79 Arthur, A.; Bearden, F. E.; Caro R. M. et al., Industrial Real-Time FORTRAN. Proposal of the Purdue Europe Technical Committee 1 on Industrial Real-Time FORTRAN, PE TC1, 2/79, March 1979.

Ada 79 Preliminary Ada Reference Manual. SIGPLAN Notices 14, 6 (1979) Part A.

AEG 76 AEG-Telefunken, Prozeßrechner AEG 80-40. Systembeschreibung. AEG-Telefunken, Frankfurt usw. A5/V.6.43/0476.

Als 71 Alsberg, Peter A., OSL/2 – An Operating System Language. PhD-Thesis, Univ. of Illinois, Urbana, 1971.

Arb 73 Arbeitsgruppe E 2 „Programmierung von Prozeßrechnern". Ein einfaches Zustandsmodell für strikt eingriffsgesteuerten Echtzeitbetrieb. Math. Inst. TU München, Bericht 7306, Mai 1973.

Bar 79 Barnes, Richard, A Working Definition Of The Proposal Extensions For PL/I Real-Time Applications. SIGPLAN-Notices 14, 10 (1979) 77. .99.

BG 71 b Bauer, F. L.; Goos, G., Informatik – Eine einführende Übersicht. 2. Teil. Springer, Berlin usw., 1971.

Bri 75c Brinch Hansen, Per, Concurrent Pascal Report. Information Science, Calif. Inst. of Techn., June 1975.

Bri 78 Brinch Hansen, Per, Distributed Processes: A Concurrent Programming Concept. CACM 21, 11 (1978) 934. .941.

BS 77a Schwald, A.; Baumann, R., PEARL im Vergleich mit anderen Echtzeitsprachen. Regelungstechnik 25, 11 (1977) 337. .344.

Bur 73 Burroughs B6700/7700 System Software Handbook. Burroughs Corp. 1973, No. 5000722.

Bur 74 Burroughs B6700/7700 ALGOL Language Reference Manual. Burroughs Corp. 20 May 1974, No 5000649.

BWW 76 Wettstein, H.; Becker-Weimann, K.; Winkler, J. F. H.; Wosnitza, H., Ein modernes, modulares Betriebssystem für Prozeßrechner und seine Generierung. Kernforschungszentrum Karlsruhe, PDV-E71, Juni 1976.

CCH 78 Clout, P. N.; Cox, A. J.; Heller, G. u. a., Industrial Real-Time FORTRAN. Purdue Europe Technical Committee 1, 1/78, Jan. 1978.

CCJ 74 Wulf, W.; Cohen, E.; Corwin, W. et al., HYDRA: The Kernel of a Multiprocessor Operating System. CACM 17, 6 (1974) 337. .345.

CHP 71 Courtois, P. J.; Heymans, F.; Parnas, D. L., Concurrent Control with "Readers" and "Writers". CACM 14, 10 (1971) 667 . . 668.

Cow 75 Cowan, Richard M., Burroughs B6700/B7700 Work Flow Language. = [Ung. 75:153 . . 166].

Dav 77 Davis, William S., Operating Systems – A systematic view. Addison-Wesley Publ. Comp., Reading, Mass., 1977.

De 71 Denning, Peter J., Third Generation Computer Systems. Computing Surveys 3, 4 (1971) 175. .216.

Dij 68a Dijkstra, E. W., Co-operating Sequential Processes. = [Ge 68:43. .112].

Dij 75 Dijkstra, Edsger W., Guarded Commands, Nondeterminacy and Formal Derivation of Programs. CACM 18, 8 (1975) 453. .457.

DM 74 Madnick, Stuart E.; Donovan, John J., Operating Systems. McGraw Hill Book Comp., New York etc., 1974.

DMN 70 Dahl, Ole-Johan; Myhrhaug, Bjørn; Nygaard, Kristen, Common Base Language. Norwegian Computing Center, Oslo, 1970.

Do 72 Donovan, John J., Systems Programming. McGraw Hill Book Comp., New York etc., 1972.

DT 76 Dorn, M.; Teuschler, G., PROZESS-FORTRAN 300 für die Siemens-Systeme 300-16 Bit. Angewandte Informatik (1976) 355. .358.

DV 66 Dennis, Jack B.; Van Horn, Earl C., Programming Semantics for Multiprogrammed Computations. CACM 9, 3 (1966) 143. .155.

Eic 75 Eichenauer, Bernd F., Dynamische Prioritätsvergabe an Tasks in Prozeßrechensystemen. Diss. Univ. Stuttgart, 1975.

FKL 76 van Wijngaarden, A.; Mailloux, B. J.; Peck, J. E. L. et al., Revised Report on the Algorithmic Language Algol 68. Springer, Berlin usw., 1976.

Ge 68 Genuys, F. (ed.), Programming Languages. Academic Press, London etc., 1968.

GMD 71 GMD-Arbeitsgruppe für Betriebssystemnormung, Abschlußbericht Teil 1: Einführung der Begriffe. St. Augustin Dezember 1971.

GS 78a Schindler, S.; Giloi, W. K. (Hrsg.), GI – 8. Jahrestagung. Springer, Berlin usw. 1978.

Gue 63 Güntsch, Fritz R., Einführung in die Programmierung digitaler Rechenautomaten. Walter de Gruyter + Co., Berlin 1963.

Haa 76 Haase, Volkmar H., Vergleichende Übersicht über Anwendungen und Implementationen von Real-Time-BASIC. = [Kro 76: 43. .54].

Hab 76 Habermann, A. N., Introduction to Operating System Design. Science Research Associates, Chicago etc. 1976.

Han 72 Hansen, Per Brinch, Short-Term Scheduling in Multiprogramming Systems. Operating System Review 6, 1–2 (1972) 101. .105.

HH 73 Heyderhoff, Peter; Hildebrand, Theodor, Informationsstrukturen – Eine Einführung in die Informatik. Bibl. Institut, Mannheim usw. 1973.

Hoa 74 Hoare, C. A. R., Monitors: An Operating System Structuring Concept. CACM 17, 10 (1974) 549. .557.

HS 77 Haase, Volkmar; Stucky, Wolffried, BASIC. Programmieren für Anfänger. Bibl. Institut, Mannheim usw. 1977.

Huf 77 Huff, Gerhard, Implementierungsbeschreibung des CONCURRENT PASCAL-Systems. Univ. Karlsruhe, Fak. f. Informatik, Bericht 7/77.

IBM 72 IBM Corp., IBM System /360 Operating System. PL/I(F) Language Reference Manual. Order No. GC28-8201-4, 5. ed. Dec. 1972.

Ind 79 Industrial Real-Time Basic – Level 1 Enhancement –. European Workshop on Industrial Computer Systems. TC2 on Industrial Real-Time Basic. IRTB-E 79/2, 9-Apr-79.

Ind 79a Industrial Real-Time Basic – Level 2 Enhancement –. European Workshop on Industrial Computer Systems. TC2 on Industrial Real-Time Basic. IRTB-E 79/3, 9-Apr-79.

ISA 78 ISA (Instrument Society of America). Draft ISA/S61.3, Industrial Computer System Fortran Procedures for the Management of Independent Interrelated Tasks. August 16, 1978.

Kam 74b Kammerer, Peter, Varianten des Monitorkonzeptes diskutiert an Anwendungsbeispielen. Univ. Karlsruhe, Fak. f. Informatik, Bericht 20/74.

Kat 73 Katzan, Harry Jr., Operating Systems. Van Nostrand Reinhold, New York etc. 1973.

Ker 78 Keramidis, Sawwas, Zur Modellierung von Betriebssystemen: eine allgemeine und effektiv implementierbare Maschine. = [NTG 78: 243..252].

Kro 76 Krönig, Dirk (Hrsg), Praxis von Sprachen, Programmiersystemen und Programmgeneratoren. Carl Hanser Verlag München usw. 1976.

Lau 78 Lauber, Rudolf, Prozeßautomatisierung und Informatik. = [Gs 78a: 381..394].

LT 79 Lewis, A.; Trainito, G., Implementation of a Standard Language for Real-Time Distributed Process Control. Proc. of the 2nd IFAC/IFIP SOCOCO, Prag 1979. Pergamon Press, 1979.

Neh 75 Nehmer, J., Dispatcher Primitives for the Construction of Operating System Kernels. Acta Informatica 5 (1975) 237..255.

NTG 78 NTG/GI Fachtagung, Struktur und Betrieb von Rechensystemen. VDE-Verlag, Berlin 1978.

PDV 77 PDV – Projekt Prozeßlenkung mit DV-Anlagen, Basic PEARL – Language description. Ges. f. Kernforschung Karlsruhe, Bericht KFK-PDV 120, 1977.

PDV 77a PDV – Projekt Prozeßlenkung mit DV-Anlagen, Full PEARL – Language description. Ges. f. Kernforschung Karlsruhe, Bericht KFK-PDV 130, 1977.

PSI 78 PSI (Hrsg), COMTEST 78. PSI – Ges. für Prozeßsteuerungs- und Informationssysteme, Berlin 1978.

Sal 66 Saltzer, J. H., Traffic Control in a multiplexed computer system. MAC-TR-30, MIT, Cambridge Mass., July 1966.

Say 71 Sayers, Anthony P. (ed.), Operating Systems Survey. Auerbach, Princeton 1971.

See 74 Seegmüller, Gerhard, Einführung in die Systemprogrammierung. Bibl. Institut Mannheim usw. 1974.

Spe 71 Sperry Rand Corp., Univac 1100 Series – Operating System. Programmer Reference. 1971.

SS 75 Staunstrup, Jørgen; Sørensen, Sven M., Platon – A High Level Language for Systems Programming. RECAU – Univ. of Aarhus, May 1975, doc. no. R-75-59.

SS 75a Sørensen, Sven M.; Staunstrup, Jørgen, Platon Reference Manual. RECAU – Univ. of Aarhus, July 1975, doc. no. R-75-58.

SW 78 Winkler, J. F. H.; Stoffel, C., Methode zur Betriebssystemgenerierung und-modularisierung für Prozeßrechensysteme. Kernforschungszentrum Karlsruhe, KfK-PDV 153, Februar 1978.

Ung 75 Unger, C. (ed.), Command Languages. North Holland, Amsterdam etc. 1975.

VDI 78 VDI/VDE 3556, Prozeß-Fortran 75. VDI/VDE Richtlinie, Entwurf März 1978.

Wal 72 Walden, David C., A System for Interprocess Communication in a Resource Sharing Computer Network. CACM 15, 4 (1972) 221..230.

Wet 78 Wettstein, Horst, Aufbau und Struktur von Betriebssystemen. Carl Hanser, München usw. 1978.

Win 78 Winkler, J. F. H., Zum Begriff des Prozesses: am Beispiel von PEARL. Elektr. Rechenanlagen 20, 6 (1978) 277..282.

Win 79 Winkler, J. F. H., Eine Theorie der Prozeßoperationen. Dissertation, Universität Karlsruhe, Juli 1979.

Wir 77 Wirth, N., Modula: a Language for Modular Multiprogramming. Software – Pract. + Exp. 7, 1 (1977) 3..35.

Wir 77c Wirth, N., Design and Implementation of Modula. Software – Pract. + Exp. 7, 1 (1977) 67..84.

Wir 78 Wirth, N., MODULA-2. Eidgenössische Technische Hochschule Zürich, Institut für Informatik. Bericht Nr. 27, Dezember 1978.

Eingegangen: 3. 12. 1979

J. F. H. Winkler
Siemens AG
D AP GE TKS
Otto-Hahn-Ring 6
D-8000 München 83