Bernd REESE, Heiko KRUMM
UNIVERSITÄT KARLSRUHE
Institut für Informatik III
Lehrstuhl für Prozeßinformatik

PRINT - Prozeßinterpreter

Übersicht:

1. Prozeßprogrammiersprachen

PRINT ist eine Prozeßprogrammiersprache, deren Eigenschaften auf heutige Mini- und Mikro-Rechner abgestimmt ist.
Um einen Vergleich mit gängigen Prozeßprogrammiersprachen ermöglichen zu können, sollen zunächst die wichtigsten Eigenschaften der drei Sprachen Realzeit-Basic, Realzeit-Fortran und Pearl aufgeführt werden.

1.1 Realzeit-Basic

Realzeit-Basic ist auf fast allen Mini- und vielen Mikrorechnern realisiert. Dies ist möglich, da geringe Systemgrößen (10-15 Kw) bei der Implementierung auf diesen Rechnern erreicht werden.
Basic ist sehr leicht erlernbar, dies wird weitgehend durch den Dialogbetrieb beim Programmaufbau und durch die einfache Sprachstruktur unterstützt. Die einfache Sprachstruktur erschwert allerdings den Aufbau hierarchisch gegliederter Programme. Insbesondere ist die Sprunganweisung (goto) nicht vermeidbar.
Für den Aufbau von Unterprogrammen existieren primitive Sprachelemente, Parameterübergabe bei Unterprogrammen ist nicht möglich.
In Basic stehen lediglich die beiden Datentypen Real und String zur Verfügung. Andere in der Prozeßdatenverarbeitung häufig gebrauchte Datentypen müssen über diese Datentypen simuliert werden.
Zur Zeit existiert noch kein einheitlicher Standard für Prozeßfunktionen.

1.2 Realzeit-Fortran

Fortran ist eine Programmiersprache, die speziell für technisch wissenschaftliche Anwendungen konzipiert wurde. Das Sprachniveau entspricht nicht mehr den heute üblichen Ansprüchen an eine höhere Programmiersprache.
Durch folgende einfache Zusätze und Erweiterungen wurde Fortran an die Erfordernisse der Prozeßdatenverarbeitung angepaßt:

- Möglichkeit zur Binärdatenverarbeitung über den Typ Integer (AND, OR, XOR sind auf den Datentyp Integer anwendbar, Zugriff auf einzelne Bits einer Integergröße möglich).
- Formaterweiterungen bezüglich Uhrzeit
- Die Prozeßperipherie wird über standardisierte externe Unterprogramme angesprochen
- Aufträge an die Ablaufsteuerung des Betriebssystems, wie z.B. zyklischer Start und Beenden eines Programms, werden ebenfalls durch standardisierte externe Unterprogramme weitergegeben
- Der Verkehr mit der Standardperipherie und der Dateiverwaltung wurde über entsprechende Unterprogramme standardisiert

Für spezielle Betriebssystemaufrufe werden von den Herstellern systemspezifische Unterprogramme zur Verfügung gestellt. Damit ist Fortran, allerdings auf Kosten der Homogenität, eine wirkungsvolle Prozeßprogrammiersprache.

1.3 Pearl

Im Gegensatz zu Realzeit-Basic und Realzeit-Fortran wurde Pearl schon von vornherein als Prozeßprogrammiersprache konzipiert. Durch die Aufteilung der Pearlprogramme in einen Systemteil und einen Problemteil wird eine weitgehende Maschinenunabhängigkeit erreicht. Anwenderprogramme lassen sich modular aufbauen. Die strukturierte Programmierung wird durch entsprechende Sprachkonstruktionen unterstützt. Alle für die Realzeitprogrammierung erforderlichen Datentypen (Binär, Real, Integer, Uhrzeit, Zeitdauer, Verbunde) sind vorgesehen.
Sprachelemente für

- Definition von Tasks (Prozesse)
- Synchronisation
- Kommunikation von Standard- u. Prozeßperipherie
- Interruptbehandlung
- Ablaufsteuerung

stehen zur Verfügung.

Da bei der Konzipierung von Pearl weitgehend keine Rücksicht auf vorhandene Betriebssysteme genommen wurde, ist eine umfangreiche Anpassung existierender Betriebssysteme an die Anforderungen der Sprache erforderlich. Dieser Grund und nicht zuletzt das hohe Sprachniveau lassen eine effektive Realisierung auf Mini- und Mikrorechnern fraglich erscheinen.

2. PRINT-Sprachbeschreibung

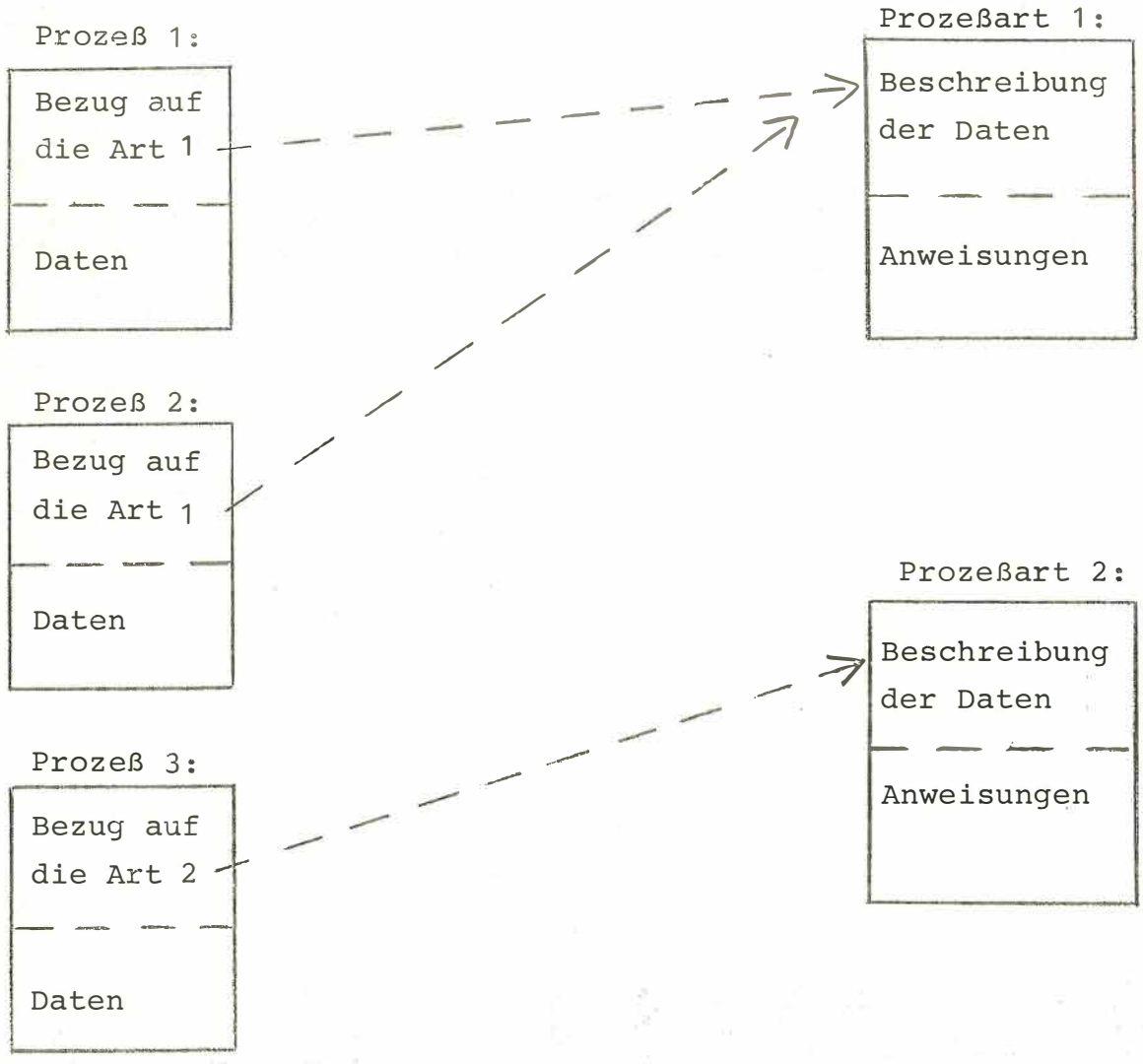
Die Sprache PRINT enthält alle notwendigen Sprachkonstruktionen einer höheren Prozeßprogrammiersprache. Trotzdem läßt die Wahl der Sprachelemente eine kompakte und effektive Realisierung auch auf kleineren Rechnern (Minis, Mikros) zu.
Dies ist insbesondere dadurch erreicht worden, daß parallel zur Sprachentwicklung bereits das dazugehörige Laufzeitsystem mit Realzeitfunktionen konzipiert wurde. Das Laufzeitsystem orientiert sich an der Bytestruktur heutiger Mini- und Mikrorechner. Außerdem wurde die Sprache PRINT so angelegt, daß Anwenderprogramme mit Hilfe eines Programmiersystems ähnlich wie bei BASIC im Dialog erstellt und verändert werden können. Insgesamt erlaubt PRINT problemnahes Programmieren, komfortable Programmerstellung und effektive Programmausführung auch bei Rechnern mit kleiner Systemkonfiguration.
Im folgenden sollen die wesentlichen Sprachelemente der Prozeßprogrammiersprache PRINT aufgeführt und erläutert werden.

2.1 Prozesse

Technische Prozesse werden in der Regel durch ihre Zustandsgrössen und deren dynamisches Verhalten beschrieben. Komplexere technische Prozesse werden durch unabhängige Teilprozesse und deren Zusammenwirken dargestellt.
In ähnlicher Weise sollten sich in einer höheren Prozeßprogrammiersprache Prozesse softwaremäßig realisieren lassen.
Zur Definition von Prozessen steht in PRINT folgende Sprachkonstruktion zur Verfügung:

```
PROCESS    <name>
  [Deklarationsteil
  [Anweisungsteil
END
```

Im Deklarationsteil werden die Zustandsgrößen des Softwareprozesses beschrieben; der Anweisungsteil beschreibt den funktionellen Ablauf des Prozesses.
Eine derartige Prozeßdefinition dient als Muster zur Erzeugung von einem oder mehreren Prozessen dieser Art. Jeder erzeugte Prozeß besteht aus einem Datenbereich und einem Bezug auf die definierte Art, in der die Daten beschrieben werden, und die zugehörige Anweisungsfolge enthalten ist.
Im nachstehenden Bild existieren zwei Prozesse der Prozeßart 1 und ein Prozeß der Prozeßart 2:



2.2 Zusammengesetzte Prozeßarten:

Da es häufig vorkommt, daß Prozeßarten gemeinsame Anweisungsfolgen und/oder gemeinsame Datendeklarationen besitzen, ist es möglich, gemeinsame Teile verschiedener Prozeßarten zu einer gemeinsamen Oberprozeßart zusammenzufassen und damit nur einmal zu programmieren.
Im folgenden Beispiel werden drei Prozeßarten U1, U2 und O beschrieben. O ist Oberprozeßart zu den Arten U1 und U2:

```
PROCESS O
[
[
END
PROCESS U1 OF O
[
[
END
PROCESS U2 OF O
[
[
END
```

2.3 Startprozeß

Jedes aus mehreren Prozeßarten bestehende Prozeßsystem muß eine Prozeßart mit der Bezeichnung MAIN enthalten. Die Bearbeitung eines Prozeßsystems beginnt mit der automatischen Erzeugung und dem Start eines Prozesses dieser Art.
Hiermit kann die Prozeßart MAIN die Rolle eines Hauptprogramms übernehmen. Alle Daten und Prozeduren des Startprozesses stehen den übrigen Prozessen zur Verfügung.

```
PROCESS MAIN
[
[
END
:
weitere Prozeßarten
:
```

2.4 Datentypen

Folgende in der Prozeßdatenverarbeitung notwendige Datentypen stehen zur Verfügung:

Integer	(Wortbreite: 16 Bit)
Real	(8 Stellen Genauigkeit durch Tetradendarstellung)
Byte	(8 Bit, für Zeichen, Binärgrößen und Zeichenketten (Strings))
Uhrzeit	(Stunde, Minute, Sekunde, Millisekunden)
Zeitdauer	(Stunden, Minuten, Sekunden, Millisekunden)
Prozeßmenge	(erlaubt den Bezug auf einen oder mehrere Prozesse)

Beispiele für Deklaration von Variablen
(Deklaration von Variablen ist obligatorisch):

INT I,J,K	Die Integervariablen I,J,K werden vereinbart
REAL(1ØØ,2ØØ)WERT	Vereinbarung eines zweidimensionalen Realfeldes (Grenzen 1ØØ und 2ØØ) mit dem Bezeichner WERT
BYTE(L) T1, T2	Dynamische Feldvereinbarung zweier eindimensionaler Bytefelder T1 und T2
CLOCK CL	Uhrzeitvariable CL
DUR D1, D2	Zeitdauervariablen D1, D2
SET REGLER	Deklaration einer Prozeßmengenvariablen mit dem Bezeichner REGLER

Beispiele für Konstantendarstellung:

Integer:	3, 1ØØ, 32767	natürliche Darstellung
	B'1ØØ1Ø1'	Binärkonstante
	O'177'	Oktalkonstante
	H'AEØØ'	Hexadezimalkonstante
Byte:	'A'	Zeichen
	'(255)','(Ø)','(1)'	Dezimaläquivalent
	'EIN TEXT'	Text
	'+-(39,1,2)*/'	Text mit Dezimaläquivalent
Real:	3.14	
	1.5 E-12	
Uhrzeit:	C'12:Ø:Ø'	12 Uhr
	TIME	Systemkonstante, aktuelle Zeit
Zeitdauer:	D'Ø:Ø:1:3Ø'	1 Sekunde 3ØØ Millisekunden
Prozeßmengen (Systemkonstanten):		
	THIS	Bezug auf aktuellen Prozeß
	NONE	Leermenge

2.5 Zuweisungen, Ausdrücke, Operatoren:

Die Schreibweise von Zuweisungen und Ausdrücken entspricht im wesentlichen der allgemein üblichen:

```
R:= X*(A+X*(B+X*C))+D
RFELD(I,J):= INTR(I+J)/10.0
```

Zusätzlich besteht die Möglichkeit, mehrere Elemente eines Feldes in einer Anweisung gleichzeitig anzusprechen.

```
TEXT(3:*):= 'XYZ', T1(1:10)
```

Text (3:*) identifiziert einen linearen Bereich des Feldes TEXT ab Element 3 mit variabler Länge.
In diesen Bereich werden die Werte der Ausdrucksliste der rechten Seite sequentiell übertragen.
Hier am Beispiel die Zeichen X, Y und Z und die ersten zehn Bytes des Feldes T1.

Monadische Operationen:

ABS SIGN ENTIER - NOT	
INTR	Explizite Artanpassung von Integer nach Real
RINT	" " " Real nach Integer
INTB	" " " Integer nach Byte
BINT	" " " Byte nach Integer
CARD	Anzahl der Elemente einer Prozeßmenge
NEW	erzeugt einen neuen Prozeß

Dyadische Operatoren:

+ - * / MOD * *	Arithmetische Operatoren
OR AND XOR	Logische Operatoren
= <> < > <= >=	Vergleichsoperatoren
CSHIFT LSHIFT	Zyklisches und logisches Schieben
LBIT RBIT	Bitleseoperatoren
CON DIS DIF MEMB	Prozeßmengenoperatoren

Prozeßmengenoperatoren:

Beispiel für das Erzeugen eines Prozesses der Art REGLER mit den Parametern P,I,D und der Priorität 5:

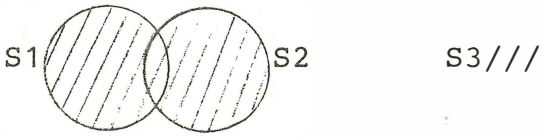
```
R1:= NEW REGLER(P,I,D) PRIO 5
```

Die Prozeßmengenvariable R1 enthält als Wert den Bezug auf den soeben erzeugten Prozeß.

Die Prozeßmengenoperatoren CON, DIS, DIF und MEMB sollen anhand von Bildern erläutert werden:

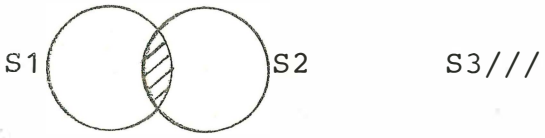
CON: Vereinigung zweier Mengen

S3:=S1 CON S2



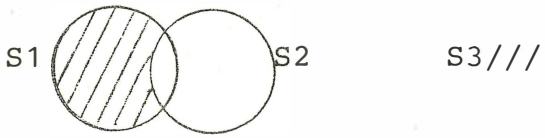
DIS: Schnittmenge

S3:=S1 DIS S2



DIF: Differenzmenge

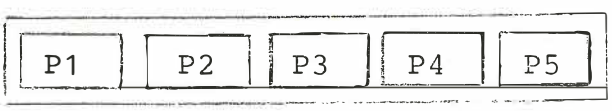
S3:=S1 DIF S2

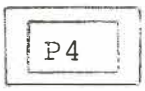


MEMB: Indizierungsoperator

Eine Prozeßmenge wird intern als lineare Liste ihrer Elemente dargestellt, mit dem MEMB Operator kann auf einzelne Elemente dieser Liste Bezug genommen werden.

S3:=S1 MEMB 4





2.6 Kontrollanweisungen

```
IF A=B  AND  C>1ØØ THEN          IF-Anweisung
  [
ELSE
  [
FI
```

```
I:=1
WHILE I<=1ØØ DO                  While-Anweisung
  FELD(I):=I**2                  (Test am Schleifenanfang)
  I:=I+1
DONE
```

```
DO                               Until-Anweisung
[                                (Test am Schleifenende)
UNTIL I=1ØØ
```

```
DO                               Endlosschleife
[
DONE
```

```
CASE I                           Verteiler
[
NEXT
[
NEXT
 :
OUT
[
ESAC
```

2.7 Unterprogramme

Deklaration und Anlauf von Unterprogrammen soll am folgenden Beispiel zur Fakultätsberechnung erläutert werden:

```
20    PROCESS MAIN
40     INT I
60     SUB FACULTAET(ARG,ERG)
80      INT ARG,ERG
100     NAME ERG
120     IF ARG<=1 THEN
140      ERG:=1
160      ELSE
180      CALL FACULTAET(ARG-1,ERG)
200      ERG:=ERG*ARG
220      FI
240     RETURN

260    COM -------- AUFRUF DES UNTERPROGRAMMS ----------
280    CALL FACULTAET(10,I)
300    END
```

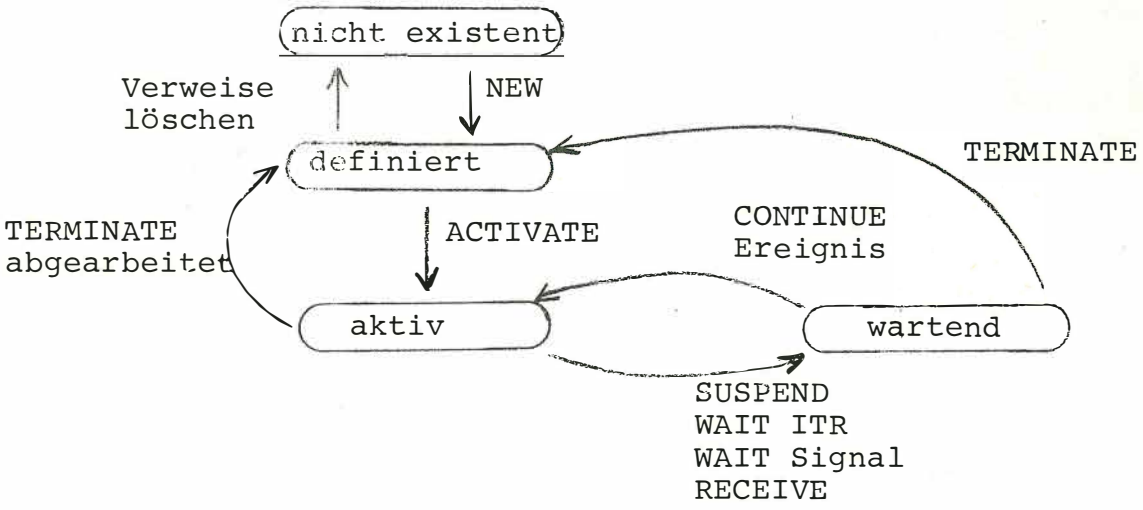
Das Unterprogramm wird durch die Anweisungen SUB und RETURN eingeschlossen. Parameter können per value oder per name übergeben werden.
Rekursiver Aufruf von Unterprogrammen ist möglich.
Unterprogramme werden mit Hilfe der CALL Anweisung aufgerufen und mit den aktuellen Parametern versehen.

Übergabe von Feldern an Unterprogramme:
Beispiel:

```
INT(1ØØ,2ØØ)FELD
:
SUB UP(F)
 INT(,)F
 :
RETURN
:
CALL UP(FELD)
```

2.8 Prozeßzustände und Übergänge

Übergangsdiagramm



Prozesse können die Zustände 'nicht existent', 'definiert', 'aktiv' und 'wartend' einnehmen.

Durch Erzeugen eines Prozesses mit dem NEW Operator gelangt er vom Zustand 'nicht existent' in den Zustand 'definiert'.

```
S1:= NEW REGLER(P,I,D) PRIO 5
```

Das Aktivieren eines definierten Prozesses wird durch die ACTIVATE Anweisung bewirkt.

```
ACTIVATE S1
```

Die SUSPEND Anweisung blockiert einen aktiven Prozeß, bis er durch eine CONTINUE Anweisung deblockiert wird.

```
SUSPEND  S1
CONTINUE S1
```

Durchläuft ein Prozeß eine WAIT ITR (Warten auf Interrupt), eine WAIT Signal (Warten auf Signal) oder eine RECEIVE (Warten auf Botschaft und Botschaft übernehmen) Anweisung, blockiert er sich selbst, bis das entsprechende Ereignis (Interrupt, Signal, Botschaft) eintrifft und ihn deblockiert.

WAIT ITR 3	Warten auf Interrupt 3
WAIT S3	Warten auf ein Signal von einem der Prozesse der Menge S3.
SIGNAL S2	Senden eines Signals an alle Prozesse der Menge S2.
RECEIVE SENDER S: A,B,C	Warten auf eine Botschaft, Übernahme der Absenderkennung in der Prozeßmengenvariablen S und der Botschaft in den Variablen A,B,C.
RECEIVE FROM S3: A,B,C	Warten auf eine Botschaft von einem der Prozesse der Menge S3, Übernahme der Botschaft in A,B,C.
SEND A,B,C TO S1	Erzeugen einer Botschaft mit den Werten von A,B,C und absenden an alle Prozesse der Menge S1.

Ist ein Prozeß abgearbeitet, d.h. er hat die letzte END Anweisung erreicht oder wird ein Prozeß durch eine TERMINATE Anweisung explizit terminiert, so gelangt er wieder in den Zustand 'definiert' und kann gegebenenfalls neu aktiviert werden.

```
TERMINATE S1
```

Wenn auf einen definierten Prozeß keine Bezugnahme mehr möglich ist, weil er in keiner Prozeßmenge des Systems mehr Mitglied ist, so wird er gelöscht und die von ihm belegten Speicherbereiche werden freigegeben.

Einplanungen

Die Anweisungen ACTIVATE, TERMINATE, SUSPEND, CONTINUE, PREVENT, SIGNAL können wie in den vorigen Beispielen sofort ausgeführt werden oder erst zu einem angegebenen Zeitpunkt nach einer angegebenen Zeitdauer oder bei Eintreten eines Interrupts:

```
ACTIVIATE S1 AT C'12:Ø:Ø'
TERMINATE S1 AFTER DAUER
CONTINUE  S1 ONITR 3
```

Durch die DELAY Anweisung können Prozesse bis zu einem Zeitpunkt oder über eine Zeitdauer blockiert werden.

```
DELAY   THIS  UNTIL C'17:Ø:Ø'
DELAY   S1    DURING D'Ø:1Ø:Ø'
```

Einplanungen können durch die PREVENT-Anweisung zurückgenommen werden.

Interruptanweisungen:
Interrupts können maskiert, demaskiert und per Anweisung simuliert werden:

```
DISABLE  5
ENABLE   I+1
TRIGGER  4
```

2.9 Externaufrufe

Über Externaufrufe können Assemblerprogrammstücke (Externroutinnen) ausgeführt werden.
Aufruf: EXTERN < Nr. der Routine >
Zum Beispiel kann die Digitalausgabe oder die Analogeingabe über Externroutinen angesprochen werden.

Beispiel für Digitalausgabe:

```
SUB  DIGAUS(WERT)
  INT WERT
  EXTERN 1
 RETURN
  :
CALL  DIGAUS(H'Ø5A1')
  :
```

Beispiel für Analogeingabe:

```
SUB  ANEIN(WERT)
 INT  WERT
 NAME WERT
 EXTERN 2
 WAIT ITR 4
 EXTERN 3
RETURN
:
CALL  ANEIN(J)
:
```

Für die Standard- und Prozeßperipherie werden dem Benutzer genormte PRINT-Treiberprozesse und Subroutinen zur Verfügung gestellt, er kann sie aber nach Bedarf erweitern und verändern.

2.10 Beispiel für ein einfaches Prozeßsystem

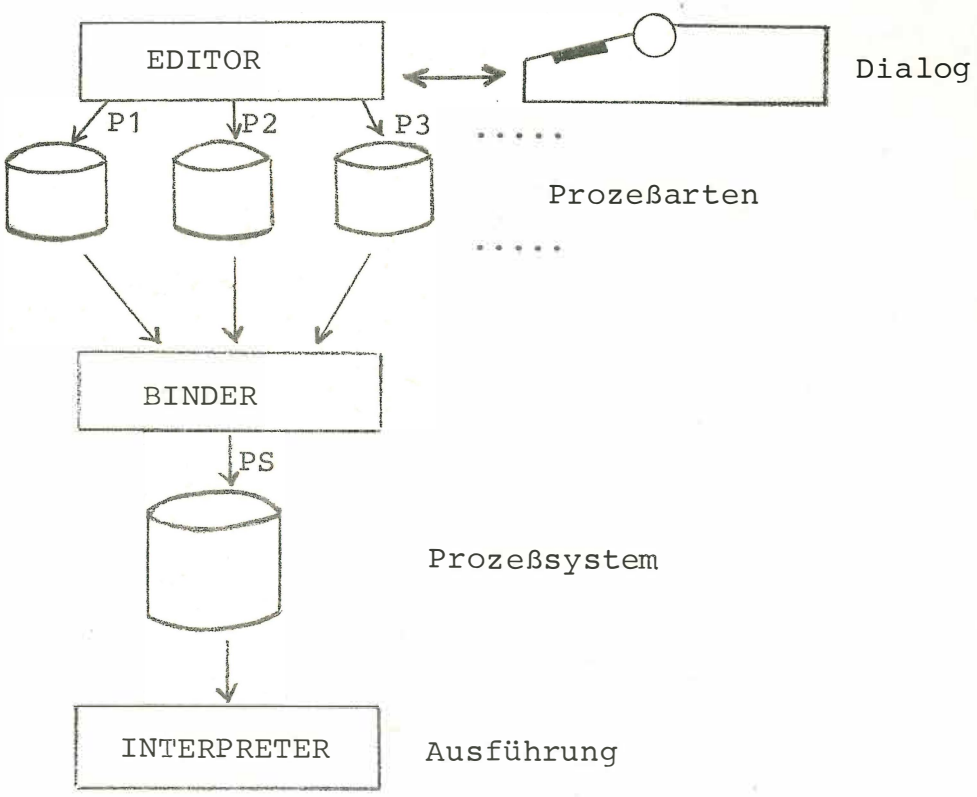
```
20    PROCESS MAIN
40      SET R
60      R:=NEW REGLER(D'0:0:1:0',6,7)
80      R:=R CON NEW REGLER(D'0:0:0:30',4,5)
100     ACTIVATE R AT C'15:0:0:0'
120     TERMINATE R AT C'17:0:0:0'
140     END


20    PROCESS REGLER(ZYKL,AS1,AS2)
40      DUR ZYKL
60      INT AS1,AS2
80      DO
100
120       COM <--- REGELALGORITHMUS --->
140
160       DELAY THIS DURING ZYKL
180       DONE
200     END
```

Der Prozeß REGLER führt einen Regelalgorithmus zyklisch mit der Zykluszeit ZYKL aus. Die Parameter AS1 und AS2 sind Schnittstellenparameter. Der Regelalgorithmus selbst ist nicht weiter ausgeführt.
Im Prozeß MAIN werden zwei Prozesse der Art REGLER mit unterschiedlichen Zykluszeiten und Schnittstellenparametern erzeugt und zu einer Prozeßmenge vereinigt. Die Prozesse dieser Menge, d.h. die beiden Regelprozesse, werden um 15.00 Uhr gestartet und um 17.00 Uhr beendet.

3. Implementierung des PRINT-Systems

3.1 Programme des Systems



3.1.1 EDITOR

Die Aufgabe des EDITOR-Programmes ist die Prozeßarterstellung im Dialog. Die Prozeßart wird zeilenweise mit Angabe der Zeilennummer aufgebaut. Jede eingegebene Zeile wird sofort auf lokale syntaktische Richtigkeit überprüft und in einen 'Analysecode' übersetzt. Es ist möglich, Zeilen gemäß Zeilennummer einzuordnen, zu überschreiben (verändern) und zu löschen. Einzelne Zeilen, Bereiche oder das ganze Programm können jederzeit am Bediengerät oder Schnelldrucker aufgelistet werden. Die Anweisungen können über das RENUM-Kommando neu nummeriert werden. Mit dem SAVE-Kommando werden Prozeßarten auf Datei abgespeichert. Hierbei wird die Struktur der Prozeßart überprüft. Auf Datei befindliche Prozeßarten können jederzeit über das OLD-Kommando wieder eingelesen und dann erneut bearbeitet werden.

3.1.2 BINDER

Die Aufgabe des Binders ist das Zusammenfügen von einzelnen Prozeßarten zu einem zusammenhängenden Prozeßsystem.
Der Binder erzeugt interpretierbaren Code.
Hierbei werden letzte semantische Überprüfungen, die nur im Zusammenhang möglich sind, durchgeführt.
Beispiel für eine Bedienung:

```
LINK  P1, P2, P3 : PS
```

3.1.3 INTERPRETER

Die Aufgabe des Interpreters ist die Interpretation des vom Binder erzeugten Codes.
Im groben läßt sich der Interpreter in folgende Funktionen aufteilen:

- Anweisungsinterpretation
- Ausdrucksinterpretation
- Organisationsteil mit
  - Hauptspeicherverwaltung
  - Interruptbearbeitung
  - Auftragsverwaltung und Bearbeitung
  - Synchronisation und Kommunikation

Da der Interpreter einen eigenen Organisationsteil besitzt, entfällt die komplizierte Anpassung an vorhandene Betriebssysteme, bzw. kann er ohne großen Aufwand als Stand-alone-System betrieben werden.

3.2 Implementierungsverfahren

Das PRINT-System wurde zunächst auf dem Großrechner UNIVAC 1108 in der höheren Programmiersprache SIMULA 67 implementiert.
Mit Hilfe dieses Systems (Editor, Binder, Interpreter in Simula) werden Editor und Binder in der Sprache Print geschrieben. Damit stehen Editor und Binder in maschinenunabhängiger Form zur Verfügung (Portabilität).
Danach wurde der Interpreter mit Organisationsteil in der Assemblersprache 300 auf dem Prozeßrechner S330 der Firma Siemens implementiert. Da Editor und Binder bereits in maschinenunabhängiger Form vorlagen, waren sie sofort auf diesem Interpreter ablauffähig.
Zur Zeit werden Interpreter für folgende Maschinen realisiert:
S330, S310, PDP11, Texas 960.