Vorstellung von polyFORTH im Einsatz

"Personal-Realtime-Computing"

Autor: Gerald Meyer
c/o RSO Gesellschaft für technische Kybernetik mbH,
Am Moosfeld 85, 8000 München 82

1. EINFÜHRUNG IN polyFORTH

Wir bezeichnen polyFORTH als ein integriertes System, weil es die normalerweise separaten Funktionen wie Betriebssystem, Editor, Compiler, Assembler, Debugger und weitere Dienstprogramme beinhaltet, die alle auf ein und derselben leistungsfähigen Sprache - dem interaktiven Forth - basieren. Daraus ergibt sich die Tatsache, daß man mit einer überschaubaren Anzahl einfacher und konsistenter Regeln den gesamten Bereich von polyFORTH beherrschen kann.

Bild 1: polyFORTH integriert die Funktionen des Betriebssystems, der Sprache, der Dienstprogramme und Applikationen.

polyFORTH wurde von vornherein mit dem ausdrücklichen Ziel entwickelt, den besonderen Anforderungen der Echtzeit-Applikation gerecht zu werden, wie schnelles Multitasking, kurze Reaktionszeiten auf Interrupts und einfache Prozesskommunikation, um nur einige zu nennen.

2. MERKMALE

Ich darf Ihnen heute die Merkmale und Methodik der Programmierung mit polyFORTH schildern.

Im Vergleich zu anderen Sprachen weist Forth folgende Merkmale auf; es ist:

- o interaktiv

  Wie Basic und Lisp ist Forth ideal um neue Ideen zu testen.

  Anders als diese Sprachen bietet es noch die weitergehende Möglichkeit spezifische Hardware ebenfalls sofort zu testen.

- o schnell

  Anders als Basic und Lisp ist Forth eine compilierte Sprache. In dieser Hinsicht ist Forth eher wie C : es ist maschinennahe.

  Außerdem besteht die Möglichkeit der unmittelbaren Programmierung von Assemblerroutinen in einer Hochsprachen-Definition.

- o strukturiert

  Wie Pascal und andere der modernen Sprachen ist Forth eine strukturierte Sprache, die damit Code von größerer Zuverlässigkeit und Logik gewährleistet.

- o erweiterbar

  Wie LOGO gestattet Forth die Schaffung einer englisch- oder deutschnahen Anwendersprache, die das spezifische Problem beschreibt.

- o modular — Wie Modula-2 unterstützt Forth die Entwicklung einzelner Software-Komponenten durch verschiedene Programmierer.

  Das führt hin zu größerer Sorgfalt bei der Codierung und beim Debugging der einzelnen Module und gewährleistet höhere Zuverlässigkeit.

Als Betriebssystem ist polyFORTH:

- o ein Multitasker — polyFORTH kann eine beliebige Anzahl konkurrierender Prozesse, die zusammen ablaufen, unterstützen. Man kann z.B. zu gleicher Zeit Daten von mehreren Laborgeräten erfassen, vorher erfaßte Daten am Terminal analysieren und einen Ausdruck erstellen.

- o ein Multiuser — polyFORTH unterstützt im Multiuserbetrieb etwa acht mal so viele Benutzer wie normalerweise UNIX. Dabei kann jeder Benutzer zusätzlich zu den gemeinsamen Routinen eigene Routinen ausführen.

- o ein Realtimer — Bei der Interrupt-Verarbeitung von polyFORTH gibt es keinen Overhead, die Intertask-Kommunikation ist höchst effizient und alle Hardware-Baugruppe werden direkt und schnell betrieben.

o flexibel — polyFORTH kann für fast jede Hardware-Zusammenstellung konfiguriert werden, egal ob es sich um ein System mit Disk oder PROM (mit sehr geringem Speicherbedarf) handelt. Das heißt auch, daß man Software sogar auf dem Zielsystem interaktiv entwickeln und testen kann.

3. METHODIK DER PROGRAMMIERUNG

Die beschriebenen Merkmale von polyFORTH führen dazu, daß die Programmierung mit polyFORTH die von der Informatik gestellten Forderungen und Erwartungen realisiert, wie

Strukturierte Programmierung

Modularität

Effizienz der SW Erstellung

Wartbarkeit der Programme.

Ich darf im Folgenden die Methodik der Programmierung mit polyFORTH erläutern.

Weil eine compilierte Forth-Definition sofort ausführbar ist, gestattet Forth den interaktiven Test und den Gebrauch kleiner Einheiten und führt damit zu einem iterativen Entwicklungsprozeß.

Bild 2 zeigt zunächst die klassische Methode des Software-Engineering, die linear vorgeht: von der Analyse zum Test am Ende. Es ist aufgrund des damit verbundenen hohen Aufwands einer Neucodierung sehr umständlich und teuer, zu irgendeinem der vorangegangenen Schritte nochmals zurückzukehren.

Bild 2: Die klassische Methode

Bei der iterativen Methode wird im Zuge der Software-Entwicklung

ein Problem analysiert,

ein Entwurf wird erstellt um den Anforderungen zu entsprechen,

der Entwurf wird implementiert und die Implementation getestet.

Jetzt werden die Testergebnisse ihrerseits analysiert, der Entwurf revidiert usw. Der Zyklus wird solange wiederholt, bis das Projekt vollständig ist.

Bild 3: Die iterative Methode

Bild 3 zeigt den Vorgang des sukzessiven "Prototypings" wie er mit der iterativen Methode von Forth unterstützt wird. Gerade diese Methode wird in allen anderen Ingenieurbereichen angewandt; sie ist gekennzeichnet von der Modellbildung, dem Studium und der Revision des Modells.

Die iterative Methode ist für die Praxis der Anwendungsprogrammierung ideal, weil dort die Spezifikationen einer ständigen Änderung unterliegen. Ein System läßt sich viel leichter bewerten, wenn es tatsächlich benutzt werden kann.

Forth wird dabei zuerst als eine Entwurfssprache benutzt, wobei die zu implementierenden Prozesse in Hochsprachen-Definitionen beschrieben werden und "Dummies" die spezifischen lower-level Prozesse emulieren. Gleichzeitig können schon Treiber für die Zielhardware entwickelt und tatsächlich auf der Hardware getestet werden. Allmählich werden in der Mitte die Details eingesetzt, während das gesamte Programm näher und näher an die Erfüllung des Gesamtziels im Projekt herangeführt wird.

Eine Software-Entwicklung dieser Art erlaubt eine sorgfältigere Überprüfung bei jedem Schritt. Z.B. kann die Benutzerschnittstelle mit simulierten Daten getestet und solange revidiert werden bis sie wirklich komfortabel und "benutzerfreundlich" ist. Auf diese Art und Weise ist die Bewertung sehr viel einfacher als durch ein Studium geschriebener Spezifikationen.

Warum nun ist in der konventionellen Programmierung das iterative Vorgehen so aufwendig? Die Antwort liegt in der Entwicklungsumgebung, die gekennzeichnet ist von separaten Funktionen wie Betriebssystemen, Editoren, Compilern etc.

Bild 4: Vergleich der Entwicklungsumgebungen

Bild 4 vergleicht den vom Programmierer zu durchlaufenden Prozess beim Gebrauch konvenioneller Sprachen wie "C" gegenüber der Methode ISD (Integrated Software Development) von polyFORTH.

Der traditionelle Programmiervorgang muß sich mit vielen separaten Modulen befassen. Jedes besitzt eigene Konventionen und Protokolle und alles benötigt sehr viel Zeit um geladen zu werden und zu laufen. Im Gegensatz dazu muß ein polyFORTH-Programmierer nur ein Programm erlernen und keine separaten Prozeduren. Sogar die Blöcke der Programm-Source, die sich in Wirklichkeit auf der Disk befinden, scheinen jederzeit speicherresident zu sein, sofort und ohne spezielle Prozeduren zugreifbar.

polyFORTH umfaßt alle Fähigkeiten, die normalerweise von einem großen Spektrum separater Programme bereitgehalten werden. Diese Integration hat mehrere wichtige Konsequenzen:

1. Der Software-Ingenieur interagiert mit einer einzigen Benutzerschnittstelle. Konventionelle Systeme enthalten jeweils separate Protokolle und Befehlssyntax für das Betriebssystem, den Compiler, den Assembler usw.

2. Die Notwendigkeit verschiedene Programme zu laden besteht nicht. Die eingesparte Zeit wie auch die vermiedene Ablenkung gestatten es, die Konzentration beim kreativen Programmierprozeß zu belassen.

3. Unterschiedliche funktionale Elemente von polyFORTH teilen Routinen für gemeinsame Vorgänge miteinander. Damit wird das System kompakt und effizient.

4. Mit diesen gemeinsamen Routinen zur Unterstützung der entwickelten Applikation wird der Programmieraufwand minimiert.

Weil eine solche Integration zur Kompaktheit führt, belegt die gesamte polyFORTH Entwicklungsumgebung normalerweise weniger als 30K Bytes. Das steht im auffallenden Gegensatz zu einer Unix/C-Umgebung, die den größten Teil einer 10 Megabyte Harddisk für die Unterbringung der verschiedenen, regelmäßig benötigten Dienstprogramme verbrauchen kann.

Bei Applikationen, die die Regelung und Steuerung spezieller Ausrüstung mit sich bringen, bringt polyFORTH spezielle Vorteile durch seine Interaktivität. Ein Programmierer, der versucht spezifische Hardware zu unterstützen, braucht die direkte Interaktion mit dieser Hardware. Das erreicht man am besten, wenn man interaktive Software auf dem Zeilsystem selbst laufen läßt.

4. PROGRAMMIERBEISPIEL

Bild 5 zeigt ein Beispiel für eine einfache Forth-Applikation. Es dreht sich um das komplette Listing einer Waschmaschinenprogrammierung und die allgemeinste, höchste Definition könnte lauten:

```
: WASCHMASCHINE   WASCHEN SCHLEUDERN SPUELEN SCHLEUDERN ;
```

Diesen Programmierstil, bei dem man die höchsten Definitionen zuerst schreibt, nennen wir Top-Down-Programmierung.

In Forth müssen Worte (Routinen) kompiliert sein, ehe man sich auf sie beziehen kann. Deshalb beginn ein Listing mit den primitivsten Definitionen und endet mit den höchsten Worten. Werden die höchsten Worte zuerst eingegeben, dann werden die niedrigen Routinen im Listing darüber editiert.

Bild 5: Programmierbeispiel WASCHMASCHINE

Im vorliegenden Beispiel werden in den Zeilen 1-2 Konstanten mit hexadezimalen Werten definiert, die Port-Adressen darstellen. In den Zeilen 5-15 definieren nacheinander die applikations-orientierten Worte, die die Funktionen ausführen.

Achten Sie bitte darauf, wie gut lesbar Forth-Code sein kann und wie leicht es ist, einen Code wie diesen zu warten. Der Code in diesem Beispiel dokumentiert sich nahezu selbst (d.h. er liest sich so gut, daß er keine Kommentare benötigt). Kommentare stehen in runden Klammern; die wenigen Kommentare, die es hier gibt, zeigen nur auf, welche Parameter für einzelne Worte übergeben werden müssen.

Sogar bei diesem sehr einfachen Beispiel wird es klar, daß Forth nicht so sehr eine Sprache, sondern eher ein Werkzeug für die Erschaffung von anwendungs-orientierten Sprachen ist. Die Definition von WASCHMASCHINE basiert nicht auf Forthworten sondern auf Worten der Waschmaschinen-Applikation wie SPUELEN und SCHLEUDERN

5. DAS BETRIEBSSYSTEM

Das polyFORTH Betriebssystem unterstützt mehrere Tasks und mehrere Benutzer, die Verwaltung von Interrupts erfolgt ohne Overhead, und es bietet direkten und schnellen Zugriff auf den Massenspeicher. Die wichtigsten Merkmale des Betriebssystems fasse ich im Folgenden zusammen.

Standalone oder Co-resident

Es handelt sich bei polyFORTH um eine komplette Entwicklungsumgebung, die als selbständiges Betriebssystem "standalone" arbeiten kann. In diesem Falle gibt es zwischen polyFORTH und der Hardware selbst keinerlei Overhead.

Wegen seiner Kompaktheit und Flexibilität kann polyFORTH aber auch auf ein anderes Betriebssystem aufgesetzt werden, mit dem es dann in der CPU "co-resident" ist. Das trifft für die Betriebssysteme CP/M, MS-DOS und RSX-11 zu.

Die meisten Merkmale von polyFORTH finden sich in beiden Versionen, dem standalone und dem co-residenten System, mit der Einschränkung, daß das schon vorhandene Betriebssystem Restriktionen für Leistung und Disk-Verwaltung mit sich bringen kann.

Speicherorganisation

Bild 6 zeigt den typischen Speicheraufbau im Mehrbenutzersystem. Die Größe der verschiedenen Benutzerbereiche kann man an die Benutzeranforderungen anpassen.

Vorkompiliertes Forth

Der Nukleus hat auf den meisten CPU's eine Größe von 8K Bytes. Es ist der einzige Systemteil, der in binärer Form vorliegt, und

wird entweder von der Diskette gebootet oder ist PROM-resident. Er enthält den Forth Compiler, Text- und Adresseninterpreter, die arithmetisch/logischen Funktionen, Disk- und Terminaltreiber, den Multiprogrammer, Editor und (meistens) den Assembler. Es sind alles re-entrante, immer verfügbare Routinen. Der gesamte Source des Nukleus ist in Forth geschrieben und kann dem Benutzer zur Verfügung gestellt werden. Mit dem Target Compiler kann man diese Source dann zur Rekompilierung von polyFORTH selbst verwenden und alle eventuellen Modifikationen entsprechend den Anforderungen der speziellen Applikation vornehmen.

Elektiven

Im Anschluß an das Power-up oder Bootstrap lädt (kompiliert) der Benutzert eine Reihe von normalerweise benötigten Routinen. Das sind solche Funktionen wie Arithmetik mit höhrer Genauigkeit, Uhr und Kalender, weitere multiprogrammierte Tasks und Applikationsroutinen, die für alle Systembenutzer zugänglich sind. Auch alle diese Routinen sind gänzlich re-entrant, und ihr Sourcecode steht dem Benutzer zur Verfügung.

Mehrere Tasks und Benutzer

Sehen Sie sich die folgenden jüngst fertiggestellten Applikationsbeispiele an:

- Ein IBM-XT auf einer Krankenhauspflegestation kommuniziert über serielle Leitungen mit 32 Terminals an den Krankenhausbetten.

- Eine große amerikanische Telefonfirma benutzt eine Datenbank mit Kunden-Fehlerprotokollen, in der eine einzige CPU mittlerweile etwa 60 Terminals unterstützt und die Eintragung von Daten und Lokalisierungs-Operationen mit einer Reaktionszeit von einer Sekunde durchführt.

polyFORTH untersstützt eine beliebige Anzahl asynchroner Prozesse, die konkurrierend ablaufen. Man nennt die Einheit, die einen dieser Prozesse ausführt eine "Task". Jede Task hat ihren eigenen Datenstack, Returnstack und Benutzervariablen. Entsprechend den Taskanforderungen können diese in unterschiedlicher Größe definiert sein; die Minimalgröße einer Task ist etwa 32 Bytes. Die Anzahl der Tasks, die definiert werden können wird nur von den Hardware-Ressourcen eingeschränkt.

Es gibt zwei Arten von Tasks:

(1) Terminal Task: mit Unterstützung einer seriellen Leitung und deshalb eines Benutzers. Sie benötigen einen größeren Benutzerbereich und bieten die Möglichkeit, für ein eigenes Benutzerlexikon.

(2) Background oder Hintergrund-Task: sind ohne Zugriff auf eine serielle Leitung und haben damit kleiner und einfachere Benutzerbereiche. Sie werden meist für die Verwaltung anderer Hardware-Funktionen verwendet.

Bild 7 zeigt, wie die Tasks in einem "Round-Robin" verkettet sind, sie haben gleiche Priorität. Der Round-Robin ist ereignisorientiert; das bedeutet eine laufende Task wird erst von der Anforderung einer I/O-Operation, einer spezifizierten Verzögerungszeit oder durch das Wort PAUSE deaktiviert. Solche Taskumschaltungen gibt es nur an den bekannten Punkten im Programmablauf, und es entfällt die Notwendigkeit, Register sicherzustellen und wiederzuholen.

Wenn eine Task beispielsweise eine I/O-Operation anfordert, dann deaktiviert sie sich selbst und wartet auf ein Interrupt als

Signal für den Abschluß der Operation. In der Zwischenzeit bedient der Round-Robin Multitasker andere Tasks in der Kette. Wenn die I/O-Operation abgeschlossen ist, dann wird die Interrupt-Routine die wartende Task beim nächsten Durchlaufen des Round-Robin wieder aktiv werden lassen.

Dieser Multitasker ist äußerst schnell; der Round-Robin Polling Zyklus benötigt exakt eine Maschinenanweisung pro Task. Auch der Vorgang der Aktivierung und Deaktivierung von Tasks ist extrem einfach.

Die von Ihnen sicherlich jetzt gestellte Frage, warum polyFORTH in seinem Multitasker keine Prioritäten benötigt, darf ich gleich nochmals explizit beantworten: ein Task-Scheduler ist für Betriebssysteme wichtig, in denen der Vorgang des "Swapping" von Tasks langsam vonstatten geht, weil hiermit versucht wird, wenigstens einige Tasks im System schnell genug zu bedienen. polyFORTH stellt die prompte Bedienung aller Tasks sicher, indem es zeitkritische Verarbeitung ohne jede Verzögerung zur Interruptzeit gestattet und indem es einen Task-Swapping Algorithmus zur Verfügung stellt, bei dem der tatsächliche Wechsel nur wenige Mikrosekunden in Anspruch nimmt. Man hat bei der Entwicklung von polyFORTH vor allem Wert darauf gelegt, durch die Betonung von Low-Overhead Operationen ein schnelles Echtzeit-Betriebssystem zu erhalten und hat nicht durch die Hinzufügung von mehr und mehr Komplexität eine high-Overhead Umgebung geschaffen.

Der Grund für die Schnelligkeit ist darin zu sehen, daß jede Task ihren eigenen Datenstack hat; es müssen keine Daten sichergestellt und zurückgeholt werden.

Ein wichtiger Punkt in der Schnelligkeit und Einfachheit des Multitaskers ist der, daß die Tasks vordefiniert sind und nicht dynamisch beim laufenden Programm zugeteilt werden. polyFORTH gewinnt seine Flexibilität im Zuteilen von Bereichen, weil die Tasks außerhalb des Nukleus definiert werden. Lockout Schutz kann vom Benutzer auf einer "wie benötigt" Basis hinzugefügt werden.

Die dahinterstehende Philosophie im polyFORTH-Design ist die, daß der Programmierer nicht einen Overhead mit sich herumtragen soll, den er in seiner speziellen Appliaktion gar nicht braucht.

Interrupt-Verwaltung ohne Overhead

polyFORFTH beinhaltet eine standardisierte Methode für die Bedienung von Hardware-Interrupts. Es wird für benutzerdefinierte und Systemroutinen jeweils dieselbe einfache Methode benutzt: die Adressen des Codeanfangs und des Interrupt-Vektors werden an das Makro INTERRUPT geliefert, das die Codeadresse in den Hardware-Vektor legt. Zwischen dem Erscheinen des Interrupts und der Ausführung des Codes liegt keinerlei Software (es sei denn die Hardware unterstützt nur einen einzigen Interrupt und die CPU muß ihn durch Polling identifizieren). Die Reaktionszeit ist unverzüglich, richtet sich nur nach der Interrupt-Latenz des Prozessors (d.h. Sicherung des Programmzählers).

Die meisten anderen Betriebssysteme, besonders die größeren belegen hier die Interrupts mit einem Durchführungsoverhead. Diese Betriebssysteme bedienen z.B. eine Interrupt-Serviceroutine, die den momentanen Hardware/ Software-Kontext sichert und die CPU einer Task zur Bedienung des Interrupts zuteilt. Wegen dieser Intervention kann dann die tatsächliche Reaktionszeit bis zu einer Millisekunde betragen.

Bild 8: Interruptverarbeitung

Bild 8 zeigt die Beziehung zwischen Interrupts und multiprogrammierten Tasks. Hier wird der Interruptcode bei Erscheinen ohne jede Verzögerung ausgeführt. er führt zeitkritische Service-Funktionen durch und setzt den Status der Task T1 auf "wake up" wenn er das nächste Mal gepolled wird um mehr komplexe, weniger zeitkritische Verarbeitung vorzunehmen.

Die Forth-Methode zur Bedienung von Interrupts verschiebt die Beziehung zwischen der Bedienung von Interupts auf Maschinenebene und auf Taskebene. Forth-Programmierer nehmen den zeitkritischen Teil des Vorgangs in eine Coderoutine auf, die direkt an den Hardware-

Interrupt angeschlossen ist. Damit wird gewährleistet, daß zeitkritische Operationen prompt durchgeführt werden. Alle Aufgaben, die nicht s o f o r t erledigt werden müssen, werden an eine Task delegiert, wo sie leicht in high-level Forth definiert werden können.

Homogene Programmierumgebung

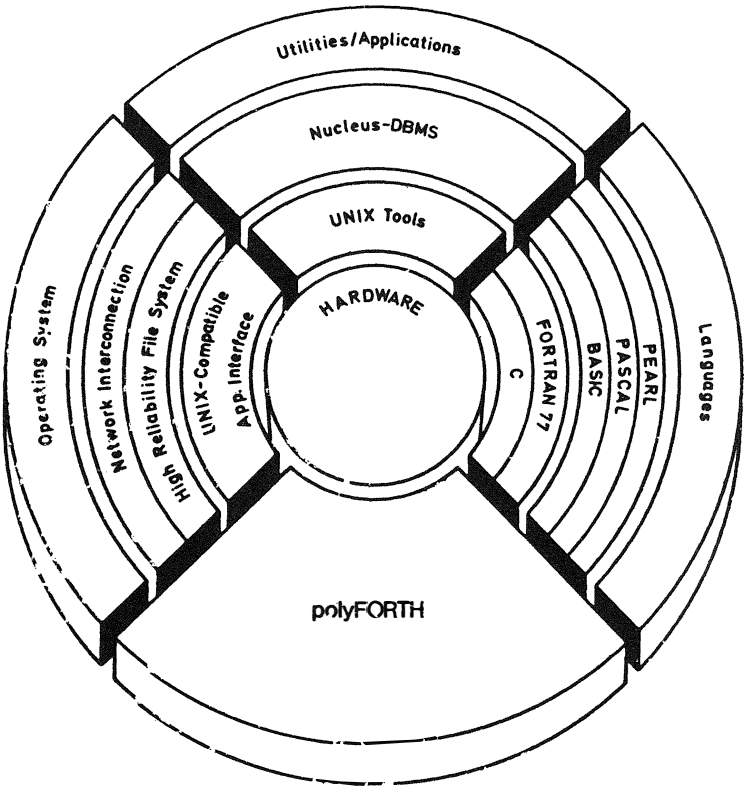

Bild 1

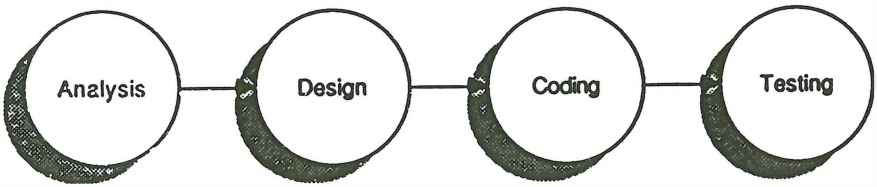
Analysis
Design
Coding
Testing

Bild 2

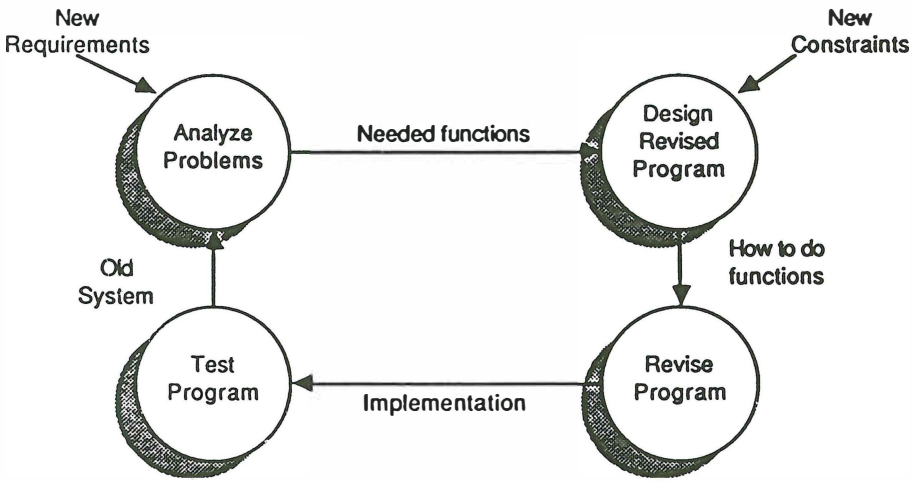
New
Requirements
New
Constraints
Analyze
Problems
Needed functions
Design
Revised
Program
How to do
functions
Old
System
Test
Program
Implementation
Revise
Program

Bild 3

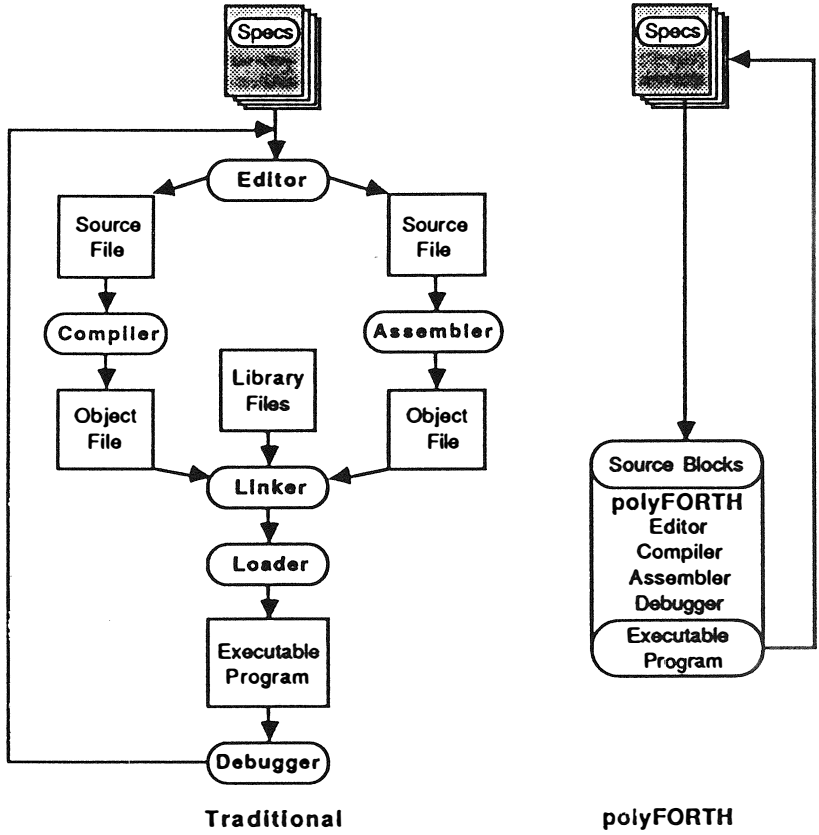
Specs
Editor
Source File
Source File
Compiler
Assembler
Library Files
Object File
Object File
Linker
Loader
Executable Program
Debugger
Traditional
Specs
Source Blocks
polyFORTH
Editor
Compiler
Assembler
Debugger
Executable Program
polyFORTH

Bild 4

```
399 LIST

     0 ( Waschmaschine Application)     HEX
     1 70 CONSTANT MOTOR    76 CONSTANT WASCHMITTEL    7A CONSTANT LEVEL
     2 72 CONSTANT PUMPE    78 CONSTANT KUPPLUNG       74 CONSTANT VENTIL
     3 DECIMAL
     4
     5 : AN ( port)  -1 SWAP OUTPUT ;  : AUS ( port)  0 SWAP OUTPUT ;
     6 : SEKUNDEN ( n)  1000 * MS ;    : MINUTEN ( n)  60 * SEKUNDEN ;
     7 : ZUGEBEN ( port)  DUP AN  10 SEKUNDEN  AUS ;
     8 : BIS_VOLL   BEGIN  LEVEL INPUT  UNTIL ;
     9 : ABPUMPEN   PUMPE AN  3 MINUTEN  PUMPE AUS ;
    10 : BEWEGEN   MOTOR AN  10 MINUTEN  MOTOR AUS ;
    11 : SCHLEUDERN   KUPPLUNG AN  BEWEGEN  KUPPLUNG AUS ;
    12 : FUELLEN   VENTIL AN  BIS_VOLL  VENTIL AUS ;
    13 : WASCHEN   FUELLEN  WASCHMITTEL ZUGEBEN  BEWEGEN  ABPUMPEN ;
    14 : SPUELEN   FUELLEN  BEWEGEN  ABPUMPEN ;
    15 : WASCHMASCHINE   WASCHEN  SCHLEUDERN  SPUELEN  SCHLEUDERN ;

400 LIST

401 LIST

  Copyright (c) RSO GmbH   22 AUG 1986   pF32-68K/VME-M Level 4
```

Bild 5

Speicheraufteilung bei polyFORTH

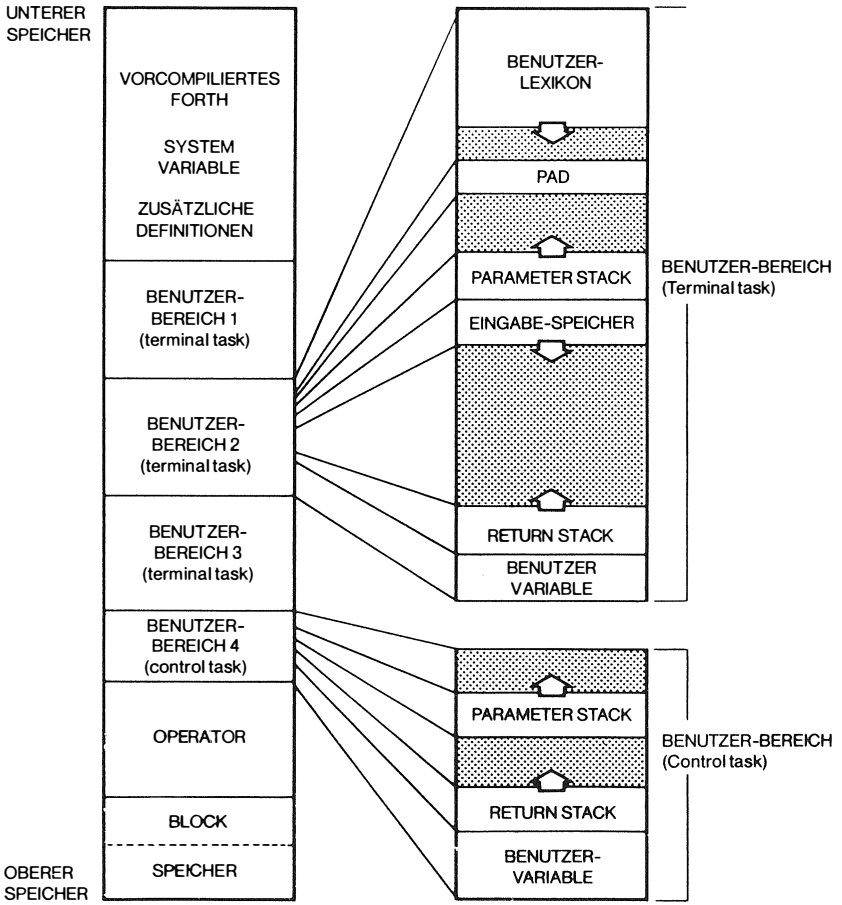

Bild 6

Multi-Programmierer Betrieb in „Round Robin“

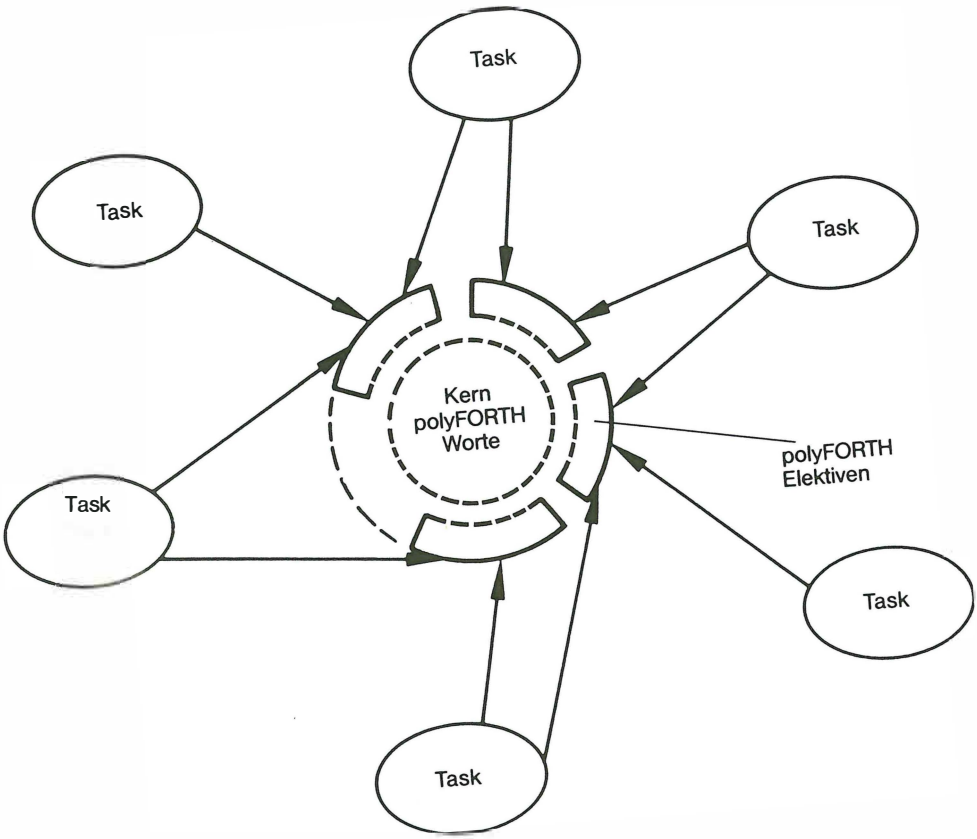

Bild 7

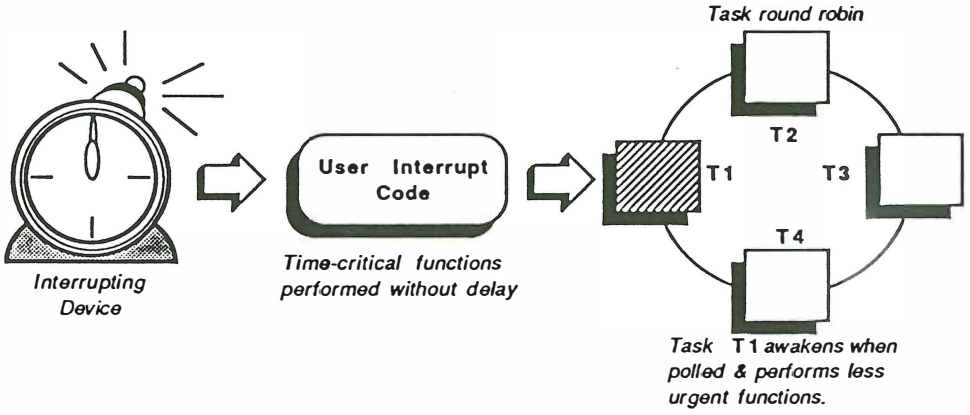
Task round robin
User Interrupt Code
T1
T2
T3
T4
Interrupting Device
Time-critical functions performed without delay
Task T1 awakens when polled & performs less urgent functions.

Bild 8