Basic-PEARL für Mini- und Mikrorechner

Prof. Dr.-Ing. W. Gerth

Die höhere Programmiersprache PEARL mit ihrem Subset Basic-PEARL [1] ist ein attraktives Instrument für die moderne Regelungstechnik. Daher wird seit einigen Jahren versucht, diese hochgradig maschinenunabhängige Sprache auch für ausgesprochene Minisysteme nutzbar zu machen. Ein solcher Ansatz soll hier vorgestellt werden. Oberstes Gebot mußte bei der Erstellung eines solchen Basic-PEARL-Systems sein, mit den Möglichkeiten eines Kleinsystems auch eine praktikable Programmentwicklung zu ermöglichen. Andererseits sollte das System nicht an einen speziellen Mini- oder Mikrorechner gebunden sein, also selbst möglichst portabel sein. Es werden im folgenden die Komponenten Betriebssystem, Compiler und Laufzeitsystem des am Institut für Regelungstechnik der Universität Hannover implementierten PEARL-Systems beschreibt

1. Sprachumfang

Der Sprachumfang umfaßt Basic-PEARL mit folgenden Beschränkungen:

- Bezeichner: max. 6 Zeichen lang
- Länge der Daten: CHAR (max.266),BIT (mac.64) FLOAT (max.56+8), FIXED (max.32)
- Signale (ON-Blöcke) noch nicht wie vorgesehen implememtiert.
- Prozeduren immer statusfrei. "Gedächtnis" muß außerhalb der Prozedur z.B. als Globale Variable angelegt sein.
- Felder können nur im IDENT-Mode an Prozeduren übergeben werden
- keine benutzerdefinierten DATIONS

Besonderheit: Alle Prozeduren sind auch ohne das Attribut REENT rekursiv und reentrant.

2. Betriebssystem

Grundlage war die VDI/VDE-Richtlinien 3554 mit einer konsequenten Trennung zwischen Prozessen 1. Art (z.B. Hardware-Interrupt-Routinen) und solchen 2.Art (Nutzer- und Systemtasks). Der Kern des Betriebssystems deckt die für Basic-PEARL erforderlichen Grundfunktionen (Multitask-Echtzeit-Manipulationen) ab, wobei für die zeitlichen Einplanungen ein "Atom" von 10 msec festgelegt wurde. Da weder der zunächst benutzte Prozessor (Mulby), noch die anvisierten 16 bit Mikros über einen geeigneten Supervisor-Call-Befehl verfügen, wurde der Dispatcher als Prozeß 1. Art mit niedrigster Priorität realisiert.
Der zugehörige Interrupt ist durch Maschinenbefehl triggerbar und wurde hardwareseitig mit einer individuellen Innenmaske versehen. Der Dispatcher durchsucht ringförmig verkettete Taskkontrollsätze und erledigt bei Taskwechsel lediglich den Austausch der 16 Register sowie von Status und Programmzähler.
Es werden auch im Endausbau nur ruhende Task ausgelagert werden können, denn die technologischen Entwicklung mit beachtlichen Adreßräumen (z.B.8086, Z 8001) macht eine Veränderung bei Suspendierung in Zukunft wahrscheinlich ohnehin überflüssig.
Die gesamte Ein/Ausgabe wurde unter Verwendung der ohnehin vorhandenen PEARL-Kernfunktionen des Betriebssystems konzipiert. Zu diesem Zweck wurde ein Speicherbereich von 2,5 Kbyte als Puffer abgezweigt, der in 128 byte-Abschnitten anfordernden Tasks zugeordnet werden kann. Die Abschnitte können zu Warteschlangen bei den einzelnen Geräten verkettet werden.Zu jedem Peripheriegerät mit begrenzter Geschwindigkeit gehört eine "Systemtask", die sich nur hinsichtlich ihrer hohen Priorität von nutzerprogrammierten PEARL-Tasks unterscheidet. Die Aufgabe einer solchen E/A-Task besteht ausschließlich in der Bearbeitung der zugeordneten Warteschlange. Die führt eine 'SUSPEND'-Operation aus, sobald das Gerät arbeitet und wird erst durch den Geräteprozeß 1. Art wieder lauffähig.

Zur Zeit laufen alle PEARL-Ausgaben ohne Wartezeit in die entsprechenden Warteschlangen hinein.
Erst wenn kein freier Abschnitt im Puffer mehr zu finden ist, wird die Auftragsgebertask zurückgetellt. Damit können sehr kurze Durchlaufzeiten erreicht werden, selbst wenn eine Task z.B. über eine langsame Teletype ausgeben muß. Weitere Erfahrungen müssen klären, ob das Risiko ungeschickter "Zustopfprogrammierung" - mit verlängerten Reaktionszeiten

für wichtige Ein/Ausgaben - für die Praxis tragbar ist.
Eine residente "Bedientask" ermöglicht ständig die Kommunikation mit dem System. Die Kommandosprache wurde - soweit möglich - dem Sprachschatz von Basic-PEARL entnommen. So sind z.B. alle Einplanungen bezüglich Zeit/Ereignis im laufenden Betrieb leicht zu ändern. Außerdem können z.B. Task-Zustandstabellen etc. ausgegeben werden und Task gestartet, angehalten, entfernt oder über einen bindenden Lader neu erzeugt werden.
Das Betriebssystem ist bei der Anpassung an beliebige Zielprozessoren der problematischste Teil - es umfaßt ca. 26 Kbyte Code, der stark prozessorabhängig ist [2].

3. Compiler

Er wurde (als normale Nutzertask) im Code einer eigens hierfür definierten virtuellen Maschine geschrieben. Diese virtuelle Maschine benutzt ein drei Byte langes Befehlswort und besitzt 2 Indexregister (16 bit) sowie ein Charakterregister (8 bit). Die meisten virtuellen Befehle dienen zur Stringbehandlung oder zu Manipulationen an den Listen bzw. Tabellen. Es sind vier Adressierungsarten (direkt, indirekt, Index 1, Index 2) definiert.

Beim Rechner Mulby 3/35 (Zweiadressmaschine mit verkürztem Code bei Registerbenutzung) werden für den kompletten Interpreter nur ca. 600 Maschinenanweisungen benötigt. Bei den Prozessoren 8086 bzw. 8001/2 werden es möglicherweise noch weniger sein. Der virtuelle Code umfaßt z.Zt. 18 Kbyte. Es wird nur ein Lauf benötigt, die Ausgabe des Compilers ist die Assemblersprache der Zielmaschine. Bemerkenswert ist hierbei, daß ein halbvirtueller Laufzeitcode generiert wird. Anders als etwa beim bekannten "p"-Code-Konzept (Pascal für Mikrorechner) werden die vorhandenen verwendbaren Maschinenanweisungen des Zielprozessors (z.B. Grundrechenarten, Sprünge etc.) direkt generiert.
Dazu wird jeder Befehl des Zielrechners in einer Tabelle des Compilers beschrieben. Der virtuelle Laufzeitcode erfaßt bis max. 4 Operanden, die als 16 bit Distanzadressen verstanden werden.

Im Compilat werden nur drei Adressierungsarten benutzt. Bei realen Indstruktionen:direkte, indizierte und indirekte (über Register) Adressierung. Letztere dient nur zum Ersatz indiziert-indirekter Adressierung.
Bei virtuellen Instruktionen: direkte, indizierte und indiziert-indirekte Adressierung. Diese Adressierungsarten sind ein Abbild der bei PEARL erforderlichen dynamischen Speichervergabe.

Auf eine anweisungsübergreifende Code-Optimierung wurde bislang verzichtet.So ist der Compiler kompakt und mit vertretbarem Aufwand an jeden Prozessor anzupassen - lediglich die zur Zeit 600 Anweisungen im Interpreter müssen geändert werden.

Die innere Struktur ist nicht nach Übersetzungsläufen, sondern nach Quellenanweisungen geordnet, d.h. lexikalische Analyse, Syntaxprüfung und semantische Analyse erfolgen Hand in Hand. Die Fehleranzeigen erfolgen eingebettet mit einem Zeiger auf die Konfliktstelle. Diese Vorgehen ist bei der Übersetzung oft nicht sackgassenfrei, wurde aber nach statistischen Gesichtspunkten hinsichtlich der Analysenversuche gestaltet.

4. Laufzeitsystem

Es handelt sich dabei um einen statusfreien gewöhnlichen Interpreter für die virtuellen Anteile am generierten Code. Zum Beispiel werden formatierte E/A-Anweisungen sowie Einplanungen wie "ALL... ACTIVATE..." etc. durch kompakte virtuelle Instruktionen erledigt. Den jeweiligen Arbeitsspeicher besorgt sich der Interpreter beim dynamischen Speicher der exekutierten Task. Prozeduren erhalten ebenfalls über den Interpreter dynamische Speicherbereiche zugewiesen. Speicherbereiche werden grundsätzlich über das Betriebssystem vergeben und mit dem Bezeichner des verantwortlichen Rechnerprozesses markiert.
Ein Indexregister des realen Prozessors wird wie ein Basisregister benutzt.Daraus ergeben sich die oben angegebenen Adressierungsarten.

Anders als z.B. bei PASCAL können bei PEARL keine Stacks für Prozedurarbeitsspeicher benutzt werden. Die Benutzung einer Prozedur durch mehrere Tasks erfolgt nämlich nicht immer "Last-In-First-Out". Das hier realisierte Konzept verketteter Speicherbereiche ermöglicht nicht nur "Wiedereintrittsfeste" sondern darüber hinaus "rekursive"Prozeduren.

5. Ergebnisse

Das vollvirtuelle (Compiler) bzw. halbvirtuelle (Laufzeit) Konzept hat sich sowohl bei der Herstellung des Systems als auch bei der Handhabung sehr gut bewährt. Der Compiler ist längst nicht so langsam wie ursprünglich vermutet. So werden für 1000 Worte produzierten Code auf dem 2 µsec Prozessor etwa 25 Sekunden Compiler-CPU-Zeit verbraucht (ohne Assemblieren). Bei Benutzung von 2 Floppy-Disks ist ein Programm von 1000 Worten Code etwa 1,5 Minuten nach Einleitung der Aktion in Exekution. Dabei hat der Prozessor noch ca. 50 % Restkapazität.

Es hat sich herausgestellt, daß der zeitliche Engpaß beim Assemblieren liegt. Darum sollte in Zukunft statt dessen eine zweite Compilerphase mit direkter Ausgabe des verschieblichen Objektcodes ergänzt werden.

Die Ackermannfunktion F (3,4) kann trotz der gegenüber PASCAL komplexeren Speicherverwaltung noch in ca. 17 sek (2µs Prozessor) berechnet werden.
Dabei kommt es zu ca. 10 000 Prozeduraufrufen, die bis zur ca. 100. Stufe rekursiv sind. In der Praxis kommen freilich solche Rekursivitäten kaum vor, so daß der gegenüber manchen PASCAL-Implementierungen schlechtere Basic-PEARL-Wert absolut tolerabel ist.

6. Schlußbemerkungen und Ausblick

Es konnte gezeigt werden, daß mit ingenieurmäßigen praktischen Mitteln in ca. 3 Jahren durch wenige studentische Arbeiten ein lauffähiges Basic-PEARL-System erstellt werden kann - ohne weiteren finanziellen Aufwand. Die erreichte "Turnaround"-Zeit liegt bei Verwendung von Mikrosystemen mit Floppy-Disks in der Größenordnung 2 bis 3 Minuten, wenn die Programme etwa 100 PEARL-Statements umfassen.
Das System wird seit Februar 1980 im regelungstechnischen Laboreinsatz erprobt. Hier hat sich gezeigt, daß Basic-PEARL für Lehre (studentische Übungen) und Forschung in der Regelungstechnik bestens geeignet ist. Umgeschriebene FORTRAN-Programme beweisen, daß auch im konventionellen Bereich die Sprache effektiv eingesetzt werden kann.
Zur Zeit wird damit begonnen, das System auf einem 16 bit Mikroprozessor zu implementieren.

Literatur

[1] PDV-Bericht KfK, PDV 160
[2] Kippe,J. Diplpmarbeit, Institut für Regelungstechnik, 1977.
[3] Rüting,H. Diplomarbeit, Institut für Regelungstechnik, 1978.