Konzepte von Betriebssystem-Komponenten

09.01.2003

PURE

Andreas Kaiser

andik@firemail.de

1 Allgemeines

1.1 Wieso wurde Pure entwickelt?

„Normale“ Betriebssysteme haben feste Schnittstellen, um einen unkomplizierten Austausch von Software unterschiedlicher Hersteller zu ermöglichen. Allerdings bringen ebendiese festen Schnittstellen Probleme mit sich. Es können entweder wichtige Fähigkeiten für eine Anwendung fehlen, oder ungenutzte liegen brach. Dies ist besonders tragisch bei eingebetteten Systemen, da dort oftmals nur wenig Platz vorhanden ist (einige Kb). Aus ebendiesem Problem ist die Betriebssystemfamilie Pure entstanden.

1.2 Was ist Pure?

Die Idee Pure’s ist es, ein Betriebssystem als eine objektorientiert implementierte Programmfamilie zu sehen. Die Objektorientierung ermöglicht eine effiziente Implementierung der hoch modularen Systemstruktur.[1]
Das Hauptaugenmerk dieser Programmiersprache liegt auf eingebetteten Systemen. Pure hat sehr feine und viele Konfigurationsmöglichkeiten. Der Entwickler kann explizit alle Eigenschaften auswählen, welche von seiner Applikation gebraucht werden und sich das passende Betriebssystem kompilieren lassen. Pure ist sozusagen ein Baukasten zur Entwicklung von Betriebssystemen.
(Wer dies selber testen will, kann dies auf der Seite www.pure-systems.de tun)

1.3 Daten zu Pure

Das PURE-System ist in C++ implementiert und auf i80x86-, sparc-, alpha-, m68k-, ppc60x-, C167-, AVR- und ARM-basierenden Plattformen portiert. Der Nucleus besteht aus über 100 Klassen, welche mehr als 600 Methoden exportieren.

2 Aufbau

2.1 Programmfamilie

Die Programmfamilien basieren auf einer so genannten „minimale Menge von Systemfunktionen“, welche die wichtigsten Abstraktionen enthält und das Grundgerüst bildet. Ausgebaut wird dies durch „minimale Systemerweiterungen“
Der Entwurf erfolgt dabei von unten nach oben, er ist jedoch zielgerichtet auf einen bestimmten Anwendungsbereich und wird damit in umgekehrter Richtung kontrolliert.
Die Idee stammt von D.L. Parnas Mitte der Siebziger Jahre.

2.2 Implementierung

Wie schon in Punkt 1.2 angesprochen, ist Pure ein „Baukasten zur Entwicklung von Betriebssystemen“. Aufgebaut ist das Ganze als eine Bibliothek, die Objektmodule enthält. Diese sind klein hinsichtlich der enthaltenen Referenzen zu Funktionen und Variablen. Dadurch können Betriebssysteme erzeugt werden, welche nur Komponenten enthalten, die von der Anwendung auch wirklich gebraucht werden. Die Vorgaben für das Betriebssystem werden also durch die Anwendung selbst festgelegt. Vorraussetzung dafür ist natürlich eine hoch modulare Systemarchitektur, welche durch das Konzept der Programmfamilien und dessen objektorientierte Implementierung geschaffen werden.
Dieses Konzept der Programmfamilien an sich schreibt keine bestimmte Art der Implementierung vor. Es bietet sich jedoch anhand der Analogie zwischen Programmfamilien und Objektorientierung an, dies auch objektorientiert zu implementieren. Die „minimale Menge von Systemfunktionen“ entspricht dann in der Objektorientierung den Basisklassen und die „minimale Systemerweiterungen“ den Ableitungen ebendieser Basisklassen. (Bild 1)

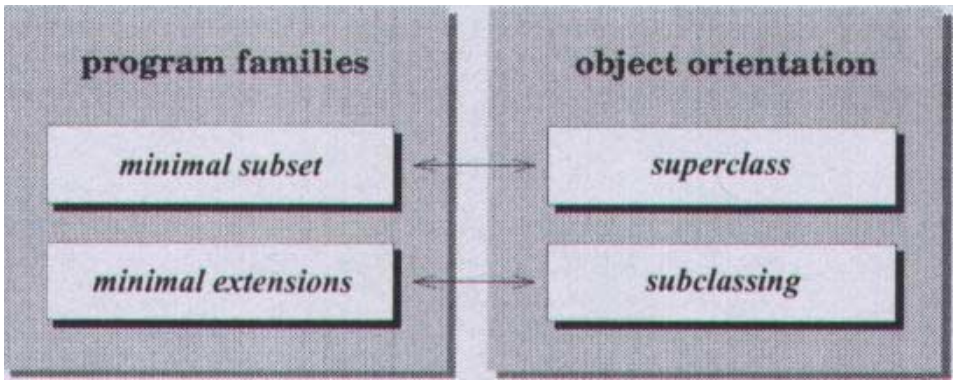

Bild 1

Realisiert wird die Implementierung weiterhin durch einen inkrementellen Systementwurf, d.h. die minimalen Systemfuntionen werden schrittweise (inkrementell) durch minimale Systemerweiterungen ergänzt.
Um kleinste wieder verwendbare Komponenten zu erhalten, sind Entwurfsentscheidungen, die die Verwendung der Komponenten einschränken würden, so weit wie möglich zurückzustellen.
Die finale Systemerweiterungen ist die Anwendung (oder mehrere Anwendungen) selbst. Die Systemkonstruktion findet also „bottom-up“ statt, wird aber „top-down“ kontrolliert. Das Betriebssystem greift ihn die Anwendung über und umgekehrt. Dadurch verschwimmen die Grenzen zwischen Betriebssystem und Anwendung.

2.3 Der Nucleus

Der Nucleus enthält mehrere Einheiten. Je nach Anforderung der Anwendung, sind die Komponenten des Nucleus unterschiedlich. Bild 2 zeigt einen Auszug aus dem Nucleus-Familienbaum. Es sind 6 Familienmitglieder zu sehen, auf die im folgenden Abschnitt genauer eingegangen wird. Jedes Mitglied besteht aus einem oder mehreren Funktionsblöcken, die in unterschiedlichen Konfigurationen wieder verwendet werden können.

2.3.1 Familienmitglieder des Nucleus

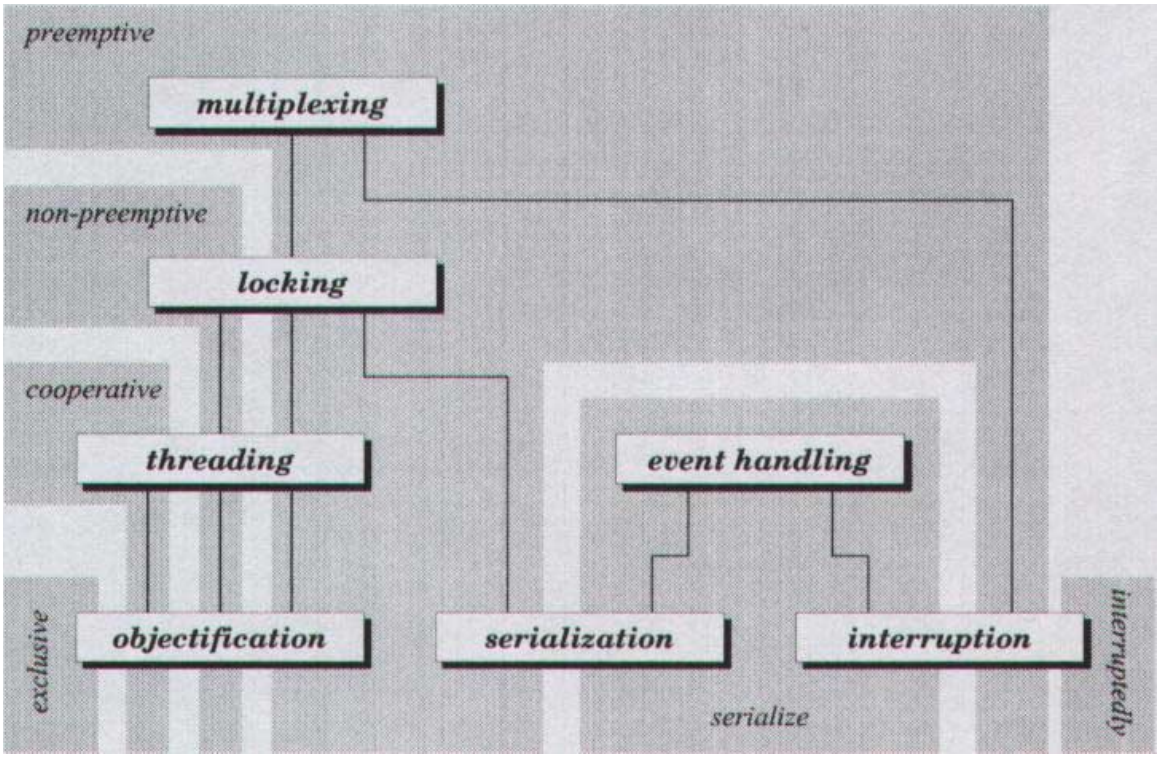

Bild 2

1. Interruption

   Diese Betriebsart verarbeitet nur Unterbrechungen. Es existieren keine Thread Abstraktionen.

2. Serialization

   Dies ist eine Erweiterung vom ersten Familienmitglied. Asynchron initiierte Aktionen der Unterbrechungsbehandlung können mit der synchronen Abarbeitung des unterbrechenden Programms abgestimmt werden.

3. Exclusive

   Diese Betriebsart kann ein einzelnes Objekt ausführen. Dies bedeutet es kann nur einen einzigen Thread geben. Das gesamte System wird durch die Anwendung kontrolliert und diese ist das einzig aktive Objekt im System.

4. Cooperative

   Threads werden durch aktive Objekte präsentiert. Der Wechsel zwischen diesen Threads wird durch die Anwendung gesteuert. Es können beliebig viele Objekte im System vorhanden sein.

5. Non- preemptive

Weiterhin kooperatives Thread-Scheaduling. Allerdings erweitert um serialisierte Ausführung (Mitglied 2), d.h. es wird die Abarbeitung der Objekte in einer interrupt-gesteuerten Umgebung ermöglicht. Der Nucleus ist allerdings in der Lage, Threads aus der Unterbrechungsbehandlung heraus zu schedulen.

6. Preemptive

Erweiterung um Module, welche ein zeitgesteuertes Thread-Scheduling übernehmen. Dies ermöglicht die autonome Abarbeitung von aktiven Objekten.

Wie im Bild 2 zu erkennen ist, bauen die Familienmitglieder 3 – 6 aufeinander auf, d.h. das Mitglied preemptive baut auf das Mitglied non-preemptive auf, usw.

2.3.2 Unterschied zwischen präemptive und nicht-präemtiv geschedulten Threads

- nicht-präemptiv:
  Ein einmal rechnender Prozess wird nicht verdrängt
- präemtiv:
  Ein rechnender Prozess kann sofort verdrängt werden falls ein neuer Prozess ankommt

3. Analyse

3.1 Speicherverbrauch

Wie die Tabelle 1 zeigt lassen sich mit Pure kleine und kompakte Systeme erzeugen. Als Basis für die Messung dienten die oben beschriebenen Mitglieder der Nucleus-Familie (Bild 2). Die Ergebnisse wurden mittels der GNU g++ 2.7.2.3 für i586 unter Red Hat Linux 5.0 ermittelt.

family member	**size** (in byte)			
	text	*data*	*bss*	*total*
interrupedly	812	64	392	1268
serialize	1882	8	416	2306
exclusive	434	0	0	434
cooperative	1620	0	28	1648
non-preemptive	1671	0	28	1699
preemptive	3642	8	428	4062

Tabelle 1

Deutlich zu sehen sind auch die Unterschiede zwischen den einzelnen Mitgliedern (im speziellen exclusive – cooperative – preemptive). Dies macht noch einmal deutlich was in Punkt 1.1 angesprochen wurde: Bei eingebetteten Systemen kann es entscheidend sein, ob ein Betriebssystem 500 oder 4000 Byte groß ist, weswegen es wichtig ist, den Platz optimal zu nutzen und keine redundanten Komponenten, welche unbenutzt bleiben ins Betriebssystem zu integrieren.

3.2 Leistungsverhalten

Die folgenden Statistiken wurden anhand eines i586 mit 166 MHz erstellt. Die Messung erfolgte mittels des Zähler-Registers des Pentium-Prozessors. Es wurde derselbe C++ Compiler wie oben in 3.1 verwendet.

Die Abarbeitung eines Interrupts beträgt je nach Konfiguration zwischen 238 und 261 Taktzyklen. Genauere Behandlung in [1] und [4].
Die Zeiten für das Schedulen der Threads hängen vom Nucleus Familienmitglied ab (Punkt 2.3.1). Sie liegen zwischen 49 Takten (kooperative) und 300 Takten (preemptive)
Dadurch wird die „leichtgewichtige“ Struktur Pures deutlich, trotz der großen Anzahl an Abstraktionen (Klassen, Module, Funktionen).

4. Probleme und Ausblick auf die Zukunft

Das momentan größte Problem an Pure ist es, die jeweils passenden Komponenten für eine Anwendung zu wählen. Es gibt zwar Regeln, wie man funktionierende Kombinationen der Module für bestimmte Anwendungen zusammenstellt, allerdings kann bis jetzt nicht formal bewiesen werden, dass diese dann auch die optimale Kombination für die jeweilige Anwendung ist. Die Lösung dieses Problems ist das Hauptziel für die Zukunft. Hierzu ein Zitat von Wolfgang Schröder-Preikschat:

„Die Eigenschaften aller in der Betriebssystemfamilie vorhandenen Komponenten (oder Abstraktionen) sind spezifiziert und bilden zusammen mit den Komponenten eine Datenbasis, aus der anhand von Anforderungsspezifikationen das für den vorgegebenen Einsatzbereich am besten passende und hinsichtlich seiner Funktion formal verifizierte Betriebssystem automatisch generiert wird. Dies ist die hinter PURE stehende Vision, die es noch weiter umzusetzen gilt und die zu einem Forschungsschwerpunkt meiner Arbeitsgruppe zählt.“[3]

Literatur:

[1] Holger Papajewski, Wolfgang Schröder-Preikschat, Olaf Spinczyk und Ute Spinczyk, "Die Pure-OSEK-API, Spezialisierung einer objektorientierten Betriebssystem-Familie"
http://ivs.cs.uni-magdeburg.de/bs/papers/profiline-incar00/pure-osek.pdf

[2] Danilo Beuche, Abdelaziz Guerrouat, Holger Papajewski, Wolfgang Schröder-Preikschat, Olaf Spinczyk, and Ute Spinczyk, "The PURE Family of Object-Oriented Operating Systems for Deeply Embedded Systems"
http://ivs.cs.uni-magdeburg.de/bs/papers/isorc99/isorc99.ps.gz

[3] Wolfgang Schröder-Preikschat, „Betriebssystembaukasten“
Grundlegendes, Geschichtliches und Ausblick
http://ddi.cs.uni-potsdam.de/HyFISCH/Spitzenforschung/SchroederPreikschat.htm

[4] Danilo Beuche, Abdelaziz Guerrouat, Holger Papajewski, Wolfgang Schröder-Preikschat, Olaf Spinczyk, and Ute Spinczyk, "On the Development of Object-Oriented Operating Systems for Deeply Embedded Systems - The PURE Project"
http://ivs.cs.uni-magdeburg.de/bs/papers/ecoop99/ecoop99.ps.gz

Weitere Links:
Pure-Systems GmbH: www.pure-systems.de
Pure Development Home Page: http://ivs.cs.uni-magdeburg.de/~pure/