

Betriebssystemtechnik

Operating System Engineering (OSE)

Einleitung



1

Das Betriebssystemdilemma

„Clearly, the operating system design must be strongly influenced by the type of use for which the machine is intended. Unfortunately it is often the case with 'general purpose machines' that the type of use cannot be easily identified; a common criticism of many systems is that in attempting to be all things to all men they wind up being totally satisfactory to no-one.“

A.M. Lister and R.D. Eager
Fundamentals of Operating Systems
Fourth Edition



© 2007 Olaf Spinczyk

2

Die Eier legende Wollmilchsau

Systeme wie Windows XP, Linux oder Solaris sind auf alle Eventualitäten vorbereitet, z.B.:

- fehlerhafte/böswillige Programme
 - präemptives *Scheduling*, Adressraumtrennung
- Mehrbenutzerbetrieb
 - Authentifizierung, Schutz von Dateien und Geräten
- vielfädige Programme, interagierende Prozesse
 - Semaphore, *Sockets*
- große/viele Programme
 - virtueller Speicher, *Swapping*, *Working Set* Konzept



© 2007 Olaf Spinczyk

3

Die Eier legende Wollmilchsau

- ein Vielzweckbetriebssystem ist für den wahrscheinlichsten Fall (den Normalfall) optimiert
- in allen Fällen, die von der künstlich definierten Norm abweichen, fallen Kosten an
- auch ungenutzte Funktionen haben einen Preis
 - Laufzeitverbrauch durch unnötige Fallunterscheidungen
 - Speicherplatzbedarf
 - erhöhte Startzeiten
 - Verschlechterung der *cache-hit* Raten
- besonders problematisch sind Eigenschaften, die sich auf viele Systemfunktionen auswirken
 - Linux 2.6 Kern: `grep EPERM` liefert **1243** Treffer!



© 2007 Olaf Spinczyk

4

Vielzwecksystem - Vielzweckfunktion

eine Analogie:

- das **Vielzwecksystem**
 - stellt Systemdienste bereit
 - versucht, allen Nutzern gerecht zu werden
 - ist Resultat vieler Kompromisse
 - verbirgt die interne Struktur (*black box*)
- die **Vielzweckfunktion**
 - stellt Teilfunktionen bereit
 - ... sonst wie oben



Vielzweckfunktion printf()

```
> uname -snrm
Linux faui48 2.6.5-7.111.19-bigsmpp i686
> echo 'main(){printf("hello, world\n");}' > hello.c
> gcc -O6 -c hello.c; gcc -static -o hello hello.o
> ./hello
hello, world
> ls -l hello*
-rwxr-xr-x 1 spinczyk i4staff 2008562 2005-04-06 17:14 hello
-rw-r--r-- 1 spinczyk i4staff 34 2005-04-06 17:13 hello.c
-rw-r--r-- 1 spinczyk i4staff 856 2005-04-06 17:14 hello.o
> size hello hello.o
   text    data    bss     dec    hex filename
367486   3156   3220  373862  5b466 hello
    36         0         0     36    24 hello.o
```



Vielzweckfunktion printf()

ist die Größe ein Linux/x86 spezifisches Phänomen?

Programm	Größe (Bytes)					
	Linux				Solaris	Windows
	i686	arm	ppc	alpha	sparc	i586
hello	203966	221998	245452	453898	183373	30935
hello.o	29	42	60	62	38	29
%	0.014	0.018	0.024	0.013	0.020	0.094

wosch, OSE 2003

- nein, hello ist auf allen Plattformen sehr groß
- innerhalb der letzten Jahre hat sich das Problem auch noch verschlimmert: **203966 zu 373862**



Spezialzweckfunktion puts()

- vermutlich kann printf() einfach zu viel
- das spezialisierte puts() sollte weniger Speicher verbrauchen ...

```
> echo 'main(){puts("hello, world");}' > hello.c
> gcc -O6 -c hello.c; gcc -static -o hello hello.o
> ./hello
hello, world
> size hello hello.o
   text    data    bss     dec    hex filename
367486   3156   3220  373862  5b466 hello
    36         0         0     36    24 hello.o
```

373862 zu 373862!



Spezieller geht's nicht: write()

- der write() *system call* sollte wirklich wenig kosten ...

```
> echo 'main(){write(1,"hello, world\n",13);}' > hello.c
> gcc -O6 -c hello.c; gcc -static -o hello hello.o
> ./hello
hello, world
> size hello hello.o
   text    data     bss      dec     hex filename
367134    3100    3252   373486   5b2ee hello
    39         0         0      39      27 hello.o
```

373862 zu 373486 :-)



Der Einfluss des Startup Code

Wieso wird der Speicherplatzverbrauch nicht signifikant kleiner?

objdump --reloc -D hello | grep printf
... liefert **1488 Treffer**. Es ist der *Startup Code*!

```
> echo '_start(){write(1,"hello, world\n",13);_exit(0);}' >
hello.c
> gcc -O6 -c hello.c; gcc -static -nostartfiles -o hello hello.o
> ./hello
hello, world
> size hello hello.o
   text    data     bss      dec     hex filename
   354         0         8     362    16a hello
    46         0         0      46     2e hello.o
```

373862 zu 362!



Hello, World - Fazit

ein typisches „hello, world“ Programm braucht unter Linux/x86 mindestens **1000 mal mehr Speicher als notwendig**. Ursachen?

- Softwarestruktur
 - Modularisierung und Offenlegung der Struktur
- Programmiersprache
 - Schnittstellen
- Werkzeugkette
 - Compiler, Objektmodule, Binder

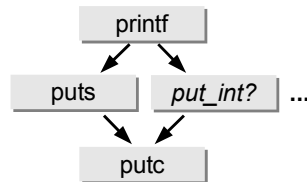


Ursachen - Softwarestruktur

- printf() unterstützt vieles:
 - alle „plain old data types“
 - [signed|unsigned][short|long] int, float, double, char, void*, char*
 - verschiedene Darstellungsformen
 - binär, oktal, dezimal, hexadezimal
 - unterschiedliche Ausrichtungen
 - rechts/links, benutzerdefinierte Feldbreiten
- nicht jede Applikation benötigt all diese *Features*
 - „Hello, World“: char*, linksbündig
- trotzdem „zahlt“ jede Applikation dafür!



Ursachen - Softwarestruktur



- aber basiert printf() auf puts()?
 - oder vielleicht puts() auf printf()?
 - oder gibt es gar keine Beziehung
- die Beziehungen zwischen den Funktionen sind nicht Teil der libc Schnittstellendefinition
 - ein Nachteil des *black box* Prinzips



Ursachen - Programmiersprache

- printf() soll eine einheitliche Schnittstelle für die formatierte Ausgabe bereitstellen.
 - Interpretation der Format-Zeichenkette zur Laufzeit
 - Referenzierung von Ausgabefunktionen für alle unterstützten Datentypen und Formate
 - keine Möglichkeit der Optimierung durch den Übersetzer, da printf() eine Bibliotheksfunktion ist
- C++ stellt mit den << Operator ebenfalls eine einheitliche Schnittstelle zur formatierten Ausgabe bereit
 - der Übersetzer ermittelt die aufgerufene Ausgabefunktion statisch
 - unbenötigte Funktionen werden nicht referenziert



Ursachen - Programmiersprache

- ist std::cout << „hello, world\n“; schlanker?

```

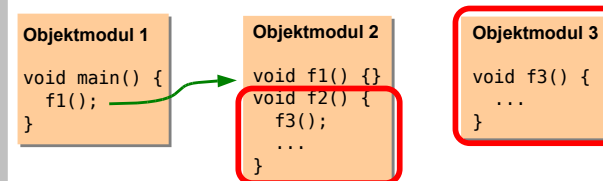
> cat > hello.cc
#include <iostream>
int main() {
    std::cout << "hello, world\n";
}
> g++ -O6 -c hello.cc; g++ -static -o hello hello.o
> ./hello
hello, world
> size hello hello.o
   text    data     bss      dec     hex filename
 781498   3952   24116  809566  c5a5e hello
    202     28     1      231    e7 hello.o
  
```

373862 zu 809566!



Ursachen - Werkzeugkette

- die GNU Werkzeugkette (gcc/g++, ar, ld) unterstützt (standardmäßig) kein Binden auf Funktionsebene



- obwohl f2() und f3() nicht referenziert werden, landen sie als unbenutzter Code im gebundenen Programm!



Ist das Problem wirklich eines?

- `printf()` und `cout` sind keine Ausnahmen
 - Programme werden immer größer, selbst wenn sie funktional nicht reicher werden
 - grundlegende Prinzipien der Softwaretechnik werden vernachlässigt

*„Some users may require only a subset of services or features other users need. These **'less demanding'** users may demand that they are not be forced to pay for the resources consumed by the unneeded features.“*

D.L. Parnas, 1979
**Designing Software for Ease
of Extension and Contraction**



Ist das Problem wirklich eines?

- wie Vielweckbetriebssysteme sind auch die GNU libc und die GNU libstdc++ für den „Normalfall“ optimiert
 - das Betriebssystem wird es schon richten
 - virtueller Speicher, *shared libraries*
 - Speicherplatzverbrauch ist im „Normalfall“ kein Problem
 - sollte man deshalb verschwenderisch damit umgehen?
- **im** Betriebssystem sieht es schon problematischer aus
 - virtueller Speicher und *shared libraries* helfen hier nicht
 - jede Anwendung hat zu leiden
- unbenutzbar wäre ein nach dem „`printf()`“ Prinzip gebautes Vielweckbetriebssystem in Domänen mit speziellen Anforderungen



Spezielle BS-Einsatzgebiete

- harte und weiche Echtzeitsysteme
 - sicherheitskritische Systeme in Fahrzeugen, Multimedia
 - deterministische Laufzeiten
- Hochleistungsrechner
 - Parallelrechner, Cluster
 - minimale Laufzeiten
- kleine Systeme
 - **eingebettete Systeme** in Massenproduktion
 - minimaler Speicherplatzverbrauch
- ... werden meist mit speziellen Betriebssystemen betrieben

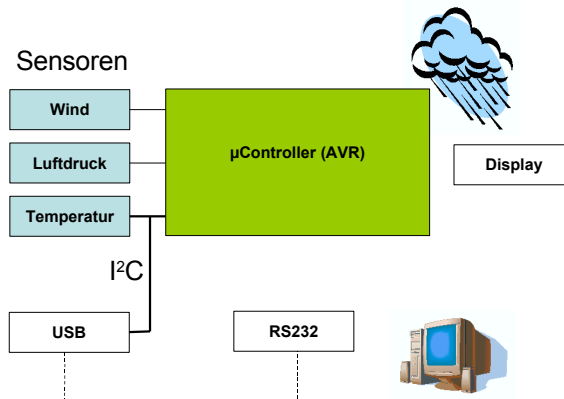


Beispiel: Eine modulare Wetterstation

- ein typisches kleines eingebettetes System
- Sensoren: Wind, Temperatur, Luftdruck, ...
- Aktoren: Display, Alarm, PC Verbindung, ...
- basiert auf einem AVR ATmega μ -controller
 - 8 Bit 4MHz RISC CPU
 - 2 – 128 kb Flash
 - 0.5 – 4 kb RAM
 - digitale, analoge, serielle und I²C basierte E/A



Beispiel: Eine modulare Wetterstation

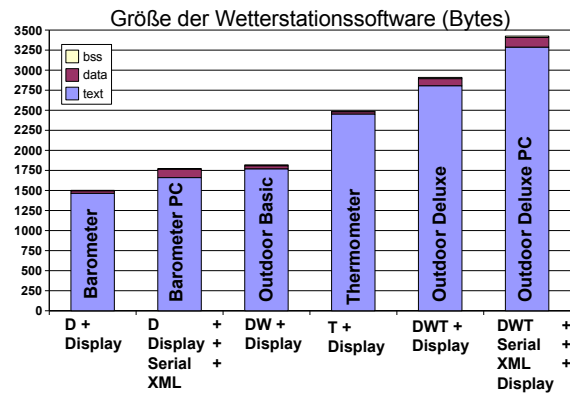


Wetterstationsvarianten

- *Thermometer:* LCD, Temperatur
- *Home:* LCD, Temperatur, Luftdruck
- *Outdoor:* LCD, Temp., Luftdruck, Wind
- *Deluxe:* + PC Verbindung
- *PC-only:* + PC Verbindung - LCD
- Serielle PC Verbindung
- USB PC Verbindung
- ...



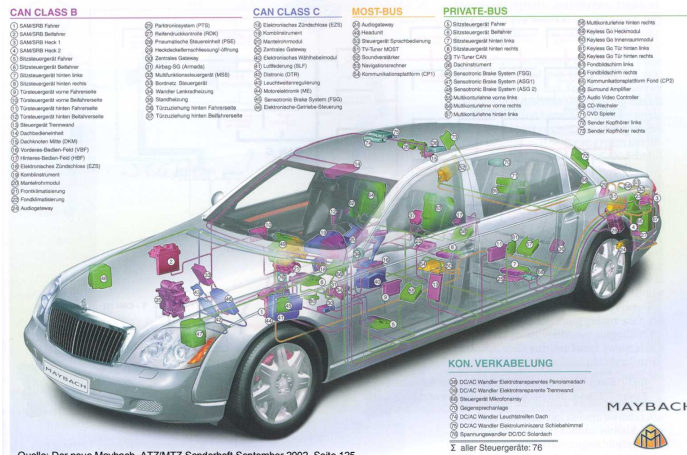
Skalierbarer Ressourcenverbrauch



Eingebettete Systeme sind überall

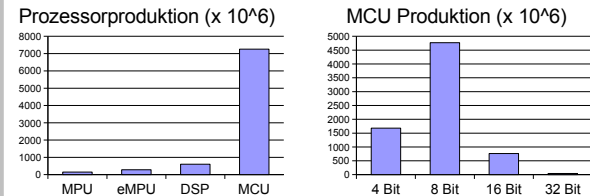


Eingebettete Systeme sind überall



Eingebettete Systeme sind überall

- eine Statistik aus dem Jahr 2000:



- von den etwa 8 Mrd. Prozessoren werden mehr als 98% im Bereich eingebetteter Systeme verwendet
- noch heute dominiert **8 Bit Technik**



© 2007 Olaf Spinczyk

26

Eingebettete Betriebssysteme?

Wie sieht ein Betriebssystem aus, das speziellen Anwendungen auf spezieller Hardware möglichst optimale Unterstützung bieten kann?

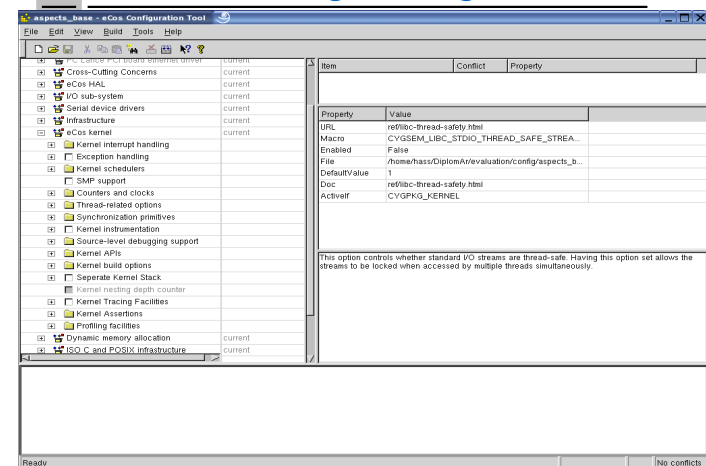
- der Markt hat vielfältige Angebote
 - ..., C{51, 166, 251}, **CiAO**, CMX RTOS, C-Smart/Raven, **eCos**, eRTOS, Embos, Ercos, Euros Plus, Hi Ross, Hynet-OS, LynxOS, MicroX/OS-II, Nucleus, OS-9, OSE, **OSEK** {Flex, Turbo, Plus}, OSEKtime, Precise/MQX, Precise/RTCS, proOSEK, pSOS, PURE, PXROS, QNX, Realos, RTMOSxx, Real Time Architect, RTA, RTX{51, 166, 251}, RTXC, Softune, SSXS RTOS, ThreadX, TinyOS, VRTX, VxWorks, ...
- über 50% des Marktes werden von proprietären Systemen abgedeckt



© 2007 Olaf Spinczyk

27

Statische Konfigurierung: z.B. eCos



Ausblick

Im Rahmen von Betriebssystemtechnik werden wir betrachten, ...

- wie die Variantenvielfalt einer „Betriebssystemfamilie“ beschreiben kann
- wie Programmfamilien strukturiert werden sollten
- welche Probleme „querschneidende Belange“ mit sich bringen und wie man sie löst
- welche modernen Konzepte zur Konfigurierung und Implementierung der Systemkomponenten existieren
- wie weit man mit der Konfigurierung gehen kann

