Flux OSKit

Vortrag im Seminar Konzepte von Betriebssystemkomponenten

Kai Selgrad
kai.selgrad@informatik.stud.uni-erlangen.de

Friedrich Alexander Universität Erlangen-Nürnberg

10. Juli 2006

Inhalt

Flux?

- Forschungsgruppe der University of Utah.
- Beschäftigt sich mit Software-Systemen:
  - Betriebssysteme,
  - Verteilte Systeme,
  - Netzwerke,
  - Compiler (“non-traditional languages”).

OSKit?

- Projekt der Flux-Gruppe.
- Bibliotheken für Betriebssystemfunktionalität:
  - Startup-Code,
  - Gerätetreiber,
  - Speicherverwaltung,
  - Minimalistische C-Bibliothek,
  - ...

Warum?

In Forschung und Lehre sind meist nur wenige spezielle Teile von Betriebssystemen interessant.
Entwickelt man ein BS komplett sind viele zusätzliche Aufgaben zu erledigen:

- boot loader,
- kernel startup code,
- device drivers,
- `printf`, `malloc`,
- ...

→ Nicht interessant unter speziellem Forschungs- bzw. Lehraspekt.

Warum nicht anders?

Übernahme von Teilen anderer Betriebsysteme generell möglich, aber:

- Betriebssysteme sehr kompliziert,
- Abhängigkeiten zwischen BS-Komponenten,
- Inkompatibilität zwischen unterschiedlichen BS.

$\longrightarrow$ Auch sehr aufwändig, zeit- und kostenintensiv.

Übliches Vorgehen

OSKit nicht aus dem Nichts entwickelt:

- Wäre sehr aufwändig,
- Bewährte Teilsysteme aus Linux/FreeBSD können übernommen werden.

Probleme:

- Inkompatible Teilsystem,
- Inderdependenzen.

Anspruch

- Bereitgestellte Komponenten müssen mit benutzerdefinierten austauschbar sein.

$\longrightarrow$ Nicht nur modular, sondern separierbar!

Beispiel: Austausch der Speicherverwaltung:
$\longrightarrow$ Wird überall verwendet!

Einen Schritt weiter

Lösungsansatz von OSKit:

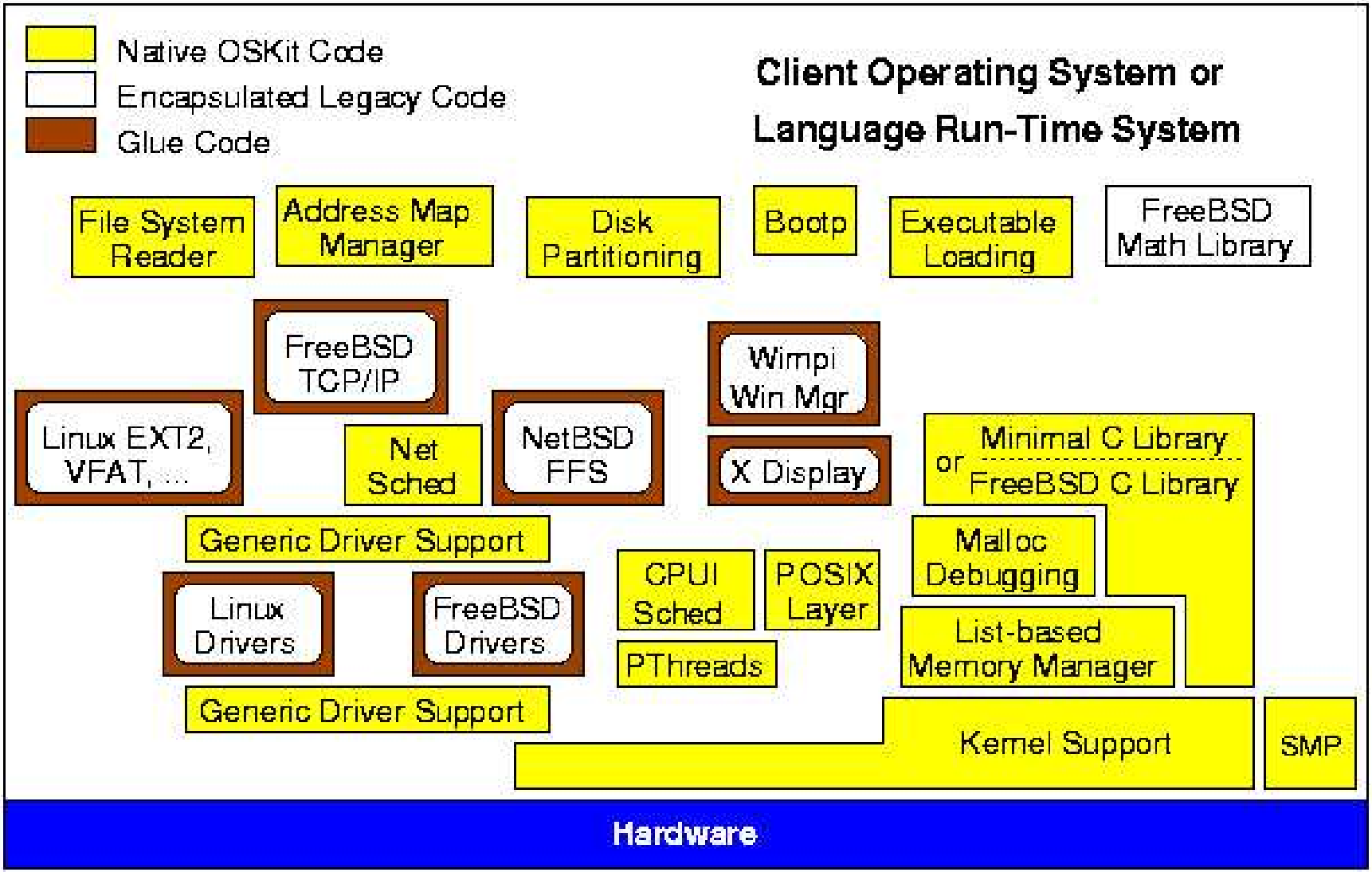
- Übernommenen (Legacy-) Code in Module gekapselt,
- Module fühlen sich in gewohnter Umgebung,
- um jedes Modul sog. Glue-Code.

→ Für den Anwender ist das Modul komplett im Glue-Code eingeschlossen.

→ Anpassung an das OSKit-Interface.

Details kommen gleich...

Bildlich


Native OSKit Code
Encapsulated Legacy Code
Glue Code
Client Operating System or
Language Run-Time System
File System Reader
Address Map Manager
Disk Partitioning
Bootp
Executable Loading
FreeBSD Math Library
FreeBSD TCP/IP
Wimpi Win Mgr
Linux EXT2, VFAT, ...
Net Sched
NetBSD FFS
X Display
Minimal C Library
or FreeBSD C Library
Generic Driver Support
Malloc Debugging
CPUI Sched
POSIX Layer
Linux Drivers
FreeBSD Drivers
List-based Memory Manager
PThreads
Generic Driver Support
Kernel Support
SMP
Hardware

COM

- Component Object Model,
- Teilsysteme stützen sich auf Interfaces,
- Zugriffe ohne Kenntnis der Repräsentation.

OSKit verwendet Teilmenge von COM um die Schnittstellen zu den einzelnen Teilsystemen einheitlich zu gestalten.
→ Glue-Code

Nochmal zusammenfassend

Legacy-Code Verwendung von Code aus verschiedenen Systemen:

- Möglichst ohne Änderungen übernehmen.
- Besonders bei Kommunikation unter Teilsystemen.

COM Einheitliche Schnittstellen zu den Komponenten:

- Aus den OS Komponenten nicht über native-, sondern über COM-Schnittstelle aufgerufen,
- Kommunikation zwischen Komponenten nur über COM sinnvoll.

$\longrightarrow$ Unabhängigkeit von spezieller Implementierung.

$\longrightarrow$ Widersprüchliche Anforderungen!

Glue-Code

Der Glue-Code kapselt den Legacy-Code auf folgende Weise:

- Aufrufe des Legacy-Codes von außerhalb erfolgen einzig über die COM-Schnittstelle.
  → Die native Schnittstelle des Legacy-Codes wird nur innerhalb der Implementierung von COM verwendet.

- Aufrufe an andere Komponenten aus dem Legacy-Code heraus werden abgefangen.
  → Die benutzte Schnittstelle wird so implementiert, dass sie die Anfrage über COM weiterleitet.

Glue-Code: Aufwand

- Beim Umleiten der zum Legacy-Code nativen Schnittstellen auf die COM-Schnittstelle müssen ggf. Datenstrukturen geändert/umgeformt werden.
  $\longrightarrow$ Kann zu zusätzlichen Kopieroperationen führen.
- Wurde soweit möglich vermieden.

Beispiele

Die Standardimplementierung der OSKit Netzwerkkomponente verwendet einen Linux Gerätetreiber und die Protokolle von FreeBSD.

- Linuxtreiber besser verfügbar,
- FreeBSD Protokolle ausgereifter.

Beispiel 1

Empfangen von Daten:

- Der Linux Netzwerktreiber empfängt ein Paket und schreibt es in einen zusammenhängenden Puffer.
- Zur Verarbeitung werden die empfangenen Daten an das Protokoll übergeben.
- Dabei gehen sie zuerst an den Glue-Code.
- Da die FreeBSD Komponente mit zusammenhängenden Puffern zurecht kommt, muss der Puffer selbst nicht kopiert werden.

Beispiel 2

Senden von Daten:

- Im Glue-Code kommen die zu sendenden Daten in möglicherweise nicht zusammenhängender Form an.
- Es wird überprüft, ob die Daten zusammenhängend sind.
- Falls ja: Direkte Weitergabe an den Linux Treiber.
- Falls nein: Umkopieren ist nötig, da der Linux Treiber nur mit zusammenhängenden Daten zurecht kommt.

Hallo Welt!

```
#include <stdio.h>
#include <oskit/clientos.h>
#include <oskit/version.h>

int main()
{
    oskit_clientos_init();

    oskit_print_version();
    printf("Hello, World\n");
    return 0;
}
```

Hallo Welt! – Erklärt (1)

oskit_clientos_init:

- Initialisiert die von OSKit bereitgestellten Komponenten.
- Das schließt u.a. die C-Bibliothek mit ein.
  → Z.B. wird die Speicherverwaltung erzeugt und an `malloc` gebunden, etc..
- Header: clientos.

oskit_print_version:

- Gibt OSKit-Version aus, mit der das BS erstellt wurde.
- Header: version.

Hallo Welt! – Erklärt (2)

printf("Hello, World\n"):

- Funktioniert wie gewohnt.
- Was nicht direkt auffällt:
  Headerdatei stdio stammt aus der C-Bibliothek von OSKit.
  → `-I$(OSKITSRC)/oskit/c/`

Fluke OS (1)

- 1996 von Grund auf neu entwickeltes Mikrokernel System.
- Zum Testen einiger Ideen im BS-Bereich.
- Ursprünglich: Basierend auf Mach Mikrokernel.
- Aber: Dieser zu groß, unflexibel und zu umfangreiche interne Abhängigkeiten.

Fluke OS (2)

Deshalb wurde ein neues System entworfen:

- Bibliothek zum Erstellen von Betriebssystemen.
  - → So sollten die Probleme, die bei Mach aufgetreten sind, vermieden werden.
- Gleichzeitiger Einsatz in verschiedenen Systemen.
  - → Sollte nicht Fluke-spezifisch werden.

Fluke benutzt nahezu die gesamte OSKit Funktionalität!

SML/NJ

Standard ML der Universität New Jersey:

- Standard ML ist eine funktionale Programmiersprache:
  - Funktionen höherer Ordnung,
  - statisches polymorphes Typensystem,
  - Exceptions,
  - Continuations, ...
- NJ Implementierung:
  - Unixbasiert,
  - ca. 144.000 Zeilen Code,
  - über 1.000 Quelltextdateien,
  - exotisches Laufzeitsystem:
    Kein Stack, statt dessen “aggressive allokation” auf dem Heap.

ML/OS

- SML/NJ wurde mit hilfe von OSKit so portiert, dass das Laufzeitsystem der Programmiersprache selbst das Betriebssystem ist.
- Von einem Master-Studenten und einer studentischen Hilfskraft innerhalb eines Semesters fertiggestellt.
- Deren angeblich hauptsächliche Beschäftigung:
  Details der SML/NJ Implementation.
- Einarbeitung in OSKit daneben vernachlässigbar.

Zusammenfassend

- Die Sache mit dem Rad.

Das war's

Danke für Eure Aufmerksamkeit

?

Noch Fragen?