Flux OSKit

Ausarbeitung zum Vortrag im Seminar Konzepte von Betriebssystemkomponenten

Kai Selgrad
sikaselg@stud.informatik.uni-erlangen.de

6. Juli 2006

Kurzzusammenfassung

OSKit ist eine Bibliothek, die bei dem Erzeugungsprozess von Betriebssystemen behilflich sein soll. Es wird ermöglicht, dass sich die Entwickler eines Betriebssystems auf die Konzepte konzentrieren können, an denen sie forschen. Die Betriebssystemkomponenten, die nicht im Zentrum des Forschens stehen, werden von dem OSKit bereitgestellt. Dieser Text befasst sich mit den Konzepten die verwendet wurden, um diese Bibliothek zu erstellen.

1 Einführung

1.1 Flux

Flux ist eine Forschungsgruppe der Universität Utah. Diese beschäftigt sich mit Softwaresystemen, wie z.B. Betriebsysteme, Verteilte Systeme, Netzwerken und Compilern (für "ungewöhnliche" Sprachen). Ein Projekt dieser Gruppe ist der im folgenden Text beschriebene OSKit.

1.2 OSKit

Der OSKit ist eine Sammlung von 34 Bibliotheken, die bei der Forschung an Betriebssystemkomponenten behilflich sein sollen. Jede dieser Bibliotheken kapselt eine bestimmte Funktionalität, die der Forscher unkompliziert übernehmen kann, wenn diese für das System zwar wichtig, für ihn aber uninteressant ist. Solche Komponenten könnten beispielsweise sein: boot loader, kernel startup code, Gerätetreiber, printf, malloc, ...

Diese Funktionalitäten könnten zwar auch aus anderen bewährten Quellen, wie z.B. Linux, FreeBSD, eigenen Projekten übernommen werden, doch kann dieser Prozess auch sehr zeitaufwändig sein, und damit an Nutzen einbüßen. Gründe dafür wären, dass Betriebssysteme sehr komplizierte Softwareprojekte sind, und man bei der Entnahme von Teilsystemen leicht Gefahr läuft, Abhängigkeiten zu übersehen. Ebenso könnten Inkompatibilitäten zwischen zwei aus verschiedenen Betriebssystemen entnommenen Teilsystemen auftreten. Somit erscheint es leichter, die vom OSKit gebotene Funktionalität zu übernehmen.

2 Zielsetzung

Zielsetzung des OSKits ist, bei Forschung und Lehre innerhalb des Betriebssystembereichs nützlich zu sein. Dies bedeutet besonders, dass das System einfach anzuwenden und zu erwei-

tern sein muss, sowie dass es möglich sein muss, Komponenten unkompliziert auszutauschen.

Die schwierigste Forderung liegt dabei wohl in der unkomplizierten Erweiterbarkeit. Es reicht also nicht, dass die einzelnen Teilsysteme modular aufgebaut sind, sie müssen auch vollständig voneinander separierbar sein. Will ein Benutzer[1] beispielsweise eine andere Speicherverwaltung benutzen, muss gewährleistet sein, dass keine Komponente Wissen über die Interna der Speicherverwaltung benutzt. Wäre dies nicht gewährleistet, würde sich das resultierende System im besten Fall unerwartet verhalten, wenn nicht vollends (vom Benutzer unverschuldet) defekt sein.

3 Konzepte

Bei der Entwicklung des OSKits waren folgende Konzepte von zentraler Bedeutung:

1. Übernahme von Quelltext aus anderen Betriebssystemen, möglichst ohne Änderungen.
2. Kooperation von Teilsystemen, die aus unterschiedlichen Betriebssystemen übernommen wurden.
3. Einheitliche Schnittstellen innerhalb von OSKit.

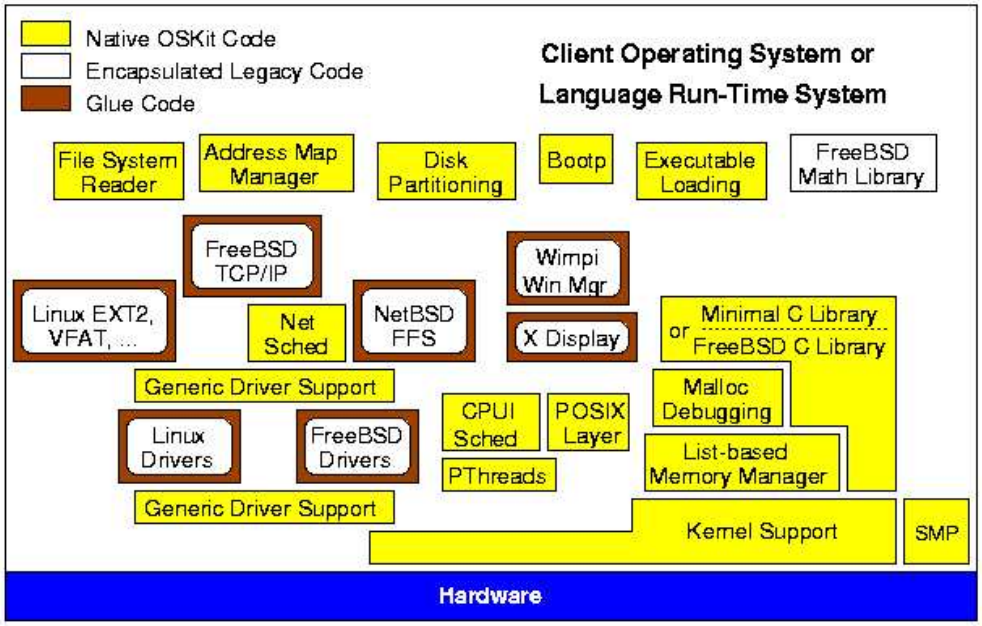

Abbildung 1: Zusammensetzung des OSKits (aus [2])

Abbildung 1 zeigt die Zusammensetzung des OSKit. Man sieht, dass sich die Implementierung stark auf bereits vorhandenen Linux- und FreeBSD-Code stützt. Im folgenden werden verschiedene Aspekte des OSKits erläutert, die zur Umsetzung der oben genannten Anforderungen und Konzepte benutzt wurden. Dabei werden auch die Bezeichnungen in Abbildung 1 klarer.

[1] Hier ist dies der Entwickler eines Betriebssystems

3.1 Legacy-Code

Bei der Implementierung des OSKits wurden große Mengen von Quellcode aus Linux und FreeBSD übernommen. Dabei spielten natürlich auch die oben genannten Probleme bei der Übernahme von Teilen anderer Systeme eine Rolle. Besonders hier waren die internen Abhängigkeiten der einzelnen Betriebssysteme problematisch. Beispielsweise wurden die Netzwerkgerätetreiber von Linux übernommen, weil diese besser verfügbar sind, aber die bewährteren FreeBSD Protokolle. Um solche Teilsysteme zusammenarbeiten zu lassen, sind größere Modifikationen an den einzelnen Quelltexten nötig, die den Nutzen dieser Wiederverwendung wie oben beschrieben relativieren.

Bei OSKit wurde ein anderer Weg gewählt. Um zu vermeiden, dass an den übernommenen Systemen viel verändert werden muss und diese dadurch möglicherweise weniger zuverlässig werden, ließ man die Systeme in ihrem ursprünglichen Zustand. Das bedeutet natürlich auch, dass sich die übernommenen Teilsysteme unverändert verhalten. Insbesondere erwarten sie, dass die Kommunikation mit anderen Teilsystemen unverändert ist. Um das zu erreichen wird den entsprechenden Systemen eine "heile Welt" insofern vorgespielt, als dass die Schnittstellen, die sie erwarten auch wirklich vorhanden sind, bzw. dass die Kommunikation über die gewohnten Datenstrukturen funktioniert. Deshalb wurde zu jedem übernommenen Teilsystem auch eine Menge von Schnittstellenfunktionen implementiert, die dafür zuständig sind, die Aufrufe und Datenstrukturen aus einer solchen Komponente heraus in eine mit dem Ziel verträgliche Form der Kommunikation umzuformen.

3.2 COM

Ein weiteres Problem besteht darin, dass Aufrufe von innerhalb des OSKits an übernommene Teilsysteme deren spezielle Kommunikationseigenschaften berücksichtigen müssen. Da auf diese Weise Abhängigkeiten von der eigentlichen Implementierung der entsprechenden Teilsysteme entstehen würden, wurden abstrakte Schnittstellen eingeführt. Diese orientieren sich an dem COM (Component Object Model) Prinzip, d.h. es existieren Schnittstellen, die an keine konkrete Implementierung gebunden sind. Um ein Objekt über eine solche Schnittstelle ansprechbar zu machen, muss es bei dieser registriert werden. Bei der Benutzug dieser Schnittstellen, wird der Aufruf an das entsprechende registrierte Objekt weitergeleitet. Dies kann man sich analog der virtuellen Funktionsaufrufe in C++ vorstellen. Dort ist es so, dass sich eine Unterklasse in eine Sprungtabelle ihrer Oberklasse einträgt. Diese Tabelle wird benutzt, um die als virtuell gekennzeichneten Funktionen anzuspringen.

Durch Anwendung dieser Schnittstellen wird der eigentliche Betriebssystemcode unabhängig von der Implementierung einzelner Teilsysteme und dadurch sehr flexibel bezüglich Änderungen, sowie leichter verständlich. Innerhalb des OSKit werden alle Module über COM angesprochen. Diese Eigenschaft ist von besonderer Wichtigkeit, wenn der Benutzer Komponenten des OSKits durch eigene ersetzt.

3.3 Glue-Code

Das Zusammenspiel von Legacy-Code[2] und COM wird als Glue-Code bezeichnet. Durch diese beiden Mechanismen werden die einzelnen Komponenten vollständig unabhängig voneinander, da sowohl die Aufrufe aus einem bestimmten Teilsystem heraus aufgefangen und angepasst werden, als auch die Ansteuerung dieser. Somit können diese Komponenten separat behandelt und ausgetauscht werden. Insbesondere kann der Benutzer Teilsysteme durch

[2]Hiermit ist insbesondere die virtuelle Umgebung gemeint, in der sich der Legacy-Code "heimisch fühlt".

eigene Implementierungen ersetzen, ohne an anderen Stellen Anpassungen vornehmen zu müssen.

Diese Flexibilität hat natürlich ihren Preis. Zu jeder Komponente, die verwendet werden soll, müssen unabhängige COM-Schnittstellen und Abfangroutinen bereitgestellt werden, die die Kommunikationsstandards und Datenstrukturen des OSKits an den Legacy-Code anpassen bzw. umgekehrt. Die dafür nötige Arbeit lohnt sich sehr wahrscheinlich aus den weiter oben aufgeführten Gründen, doch die Ausführungsgeschwindigkeit leidet unter diesem Mechanismus. Um diese Schwachstelle möglichst gering zu halten wurde – wo es ohne das Aufbrechen der Kapselung möglich war – versucht, auf Implementationsdetails Rücksicht zu nehmen. Beispielsweise werden Puffer nur dann kopiert, wenn es zwischen deren Darstellung innerhalb der beteiligten Teilsysteme zu große Inkompatibilitäten gibt (es also nicht ausreichend war, Zeiger zu kopieren).

3.4 Beispiele

Als Beispiel für die Zusammensetzung des OSKit, und der Funktionalität des Glue-Codes wird hier die Funktionsweise der OSKit Netzwerkkomponenten dargestellt. Die Standardimplementierung der Netzwerkkomponenten benutzt Linux Gerätetreiber, weil diese besser verfügbar sind, aber die Protokolle von FreeBSD, die für ausgereifter befunden wurden. Im folgenden wird beschrieben, wie das Senden bzw. das Empfangen von Daten über diese Komponente abläuft. An diesem Beispiel sieht man auch in wie weit sich ein Overhead wegen des Kopierens von Puffern vermeiden lässt – und in wie weit nicht.

Am Ende dieses Abschnittes findet sich noch ein Beispiel, das zeigt in welchem Ausmaß eine Arbeitsersparnis durch die Verwendung von OSKit ermöglicht wird.

3.4.1 Empfangen von Daten

Beim Empfangen von Daten über eine Netzwerkverbindung kommt es zu folgenden Schritten:

- Der Linux Netzwerktreiber empfängt ein Paket und schreibt es in einen zusammenhängenden Puffer.
- Bei der Übergabe an das Protokoll ruft der Linuxtreiber eine Funktion auf, die sich im Glue-Code befindet.
- An dieser Stelle werden die Datenstrukturen von Linux an die OSKit Schnittstelle angepasst. Der Puffer wird dabei nicht kopiert, der Zeiger auf die Daten wird übernommen.
- Nun wird das FreeBSD Protokoll über die COM Schnittstelle aufgerufen. Dies führt zuerst in eine Schnittstellenfunktion, welche die OSKit Repräsentation der Daten an die von FreeBSD anpasst. Hierbei wird der Zeiger auf die gelesenen Daten wieder einfach übernommen.

Beim Empfangen von Daten ergibt sich demnach nur ein geringer Mehraufwand, da ein aufwändiges Kopieren der empfangenen Daten verhindert werden kann.

3.4.2 Senden von Daten

Das Senden von Daten über einen Netzwerkverbindung läuft folgendermaßen ab:

- Um Daten zu versenden ruft das FreeBSD Protokoll den Netzwerkkartentreiber auf. Die zu sendenden Daten sind nicht zwingend in einem zusammenhängenden Puffer abgelegt, sondern können auch auf verkettete Puffer aufgeteilt sein.

- Der Aufruf des Netzwerktreibers (aus dem FreeBSD-Protokoll heraus) führt wieder in den Glue-Code.
- Dort werden die Daten an die OSKit Datenstrukturen angepasst. Da OSKit intern mit zusammenhängenden Puffern arbeitet, muss für die zu sendenden Daten ein ausreichend großes Stück Speicher allokiert und die Daten dort hineinkopiert werden.
- Der Aufruf zum Senden dieses Puffers mittels der COM Schnittstelle landet wiederum in einer Glue-Code-Funktion, die die OSKit Darstellung an die Linux Repräsentation anpasst. Hierbei wird der Zeiger auf den vorher angeforderten Speicher kopiert.

Wäre die interne OSKit Darstellung dieses Puffers an FreeBSD angepasst, so müsste spätestens im Glue-Code zum Linux Netzwerktreiber eine Kopieroperation erfolgen. Beim Senden von Daten über ein Netzwerk existiert demnach ein Mehraufwand, der ohne Änderungen am Legacy-Code unvermeidlich ist.

3.4.3 Quelltext

Im Folgenden ist der komplette zu schreibende Quelltext für das "Hallo Welt Betriebssystem" angegeben. Alle nötigen Vorbereitungen übernehmen hier von OSKit bereits implementierte Funktionen (inklusive C-Bibliothek). Die Menge an nötigem Quelltext ist natürlich stark von dem Ziel abhängig, das erreicht werden soll. Dieses Beispiel soll nur illustrieren, dass es sehr leicht ist, die umfangreichen Funktionen des OSKit zu benutzen.

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

int main() {
   oskit_clientos_init();
   oskit_print_version();
   printf("Hello, World\n");
   return 0;
}
```

Hier nur ein paar wenige Anmerkungen, für eine ausführliche Beschreibung des angegebenen Quelltextes sei an dieser Stelle auf die Dokumentation verwiesen:

oskit_clientos_init initialisiert die von OSKit bereitgestellten Komponenten. Das schließt beispielsweise die C-Bibliothek mit ein. D.h. es wird u.a. die Speicherverwaltung erzeugt und an `malloc` gebunden, etc.. (Header: clientos)

oskit_print_version gibt die OSKit-Version mit der das Betriebssystem erstellt wurde auf dem Bildschirm aus. (Header: version)

printf("Hello, World\n") funktioniert wie gewohnt. Was nicht direkt auffällt: Die Headerdatei stdio stammt aus der C-Bibliothek von OSKit (`-I$(OSKITSRC)/oskit/c/`).

4 Anwendung

Der OSKit wurde bei vielen verschiedenen Projekten verwendet. Im Folgenden werden 2 verschiedene Anwendungen kurz vorgestellt, um einen Einblick in die Anwendungsmöglichkeiten zu geben.

4.1 Fluke OS

Im Jahr 1996 wurde von der Flux-Gruppe ein von Grund auf neues mikrokernelbasiertes Betriebssystem namens Fluke entwickelt. Es sollte dazu dienen, einige neue Ideen in der Betriebssystemtechnik zu testen. Ursprünglich sollte sich diese Arbeit auf den Mach Mikrokernel stützen, welcher sich jedoch als zu groß, unflexibel und mit zu umfangreichen internen Abhängigkeiten erwies.

Aus diesen Gründen wurde ein neues System entworfen. Da die bei Mach aufgetretenen Probleme vermieden werden sollten, kam die Idee auf, die einzelnen Komponenten als Bibliotheken zu entwickeln. (OSKit wurde nebenbei noch in anderen Projekten eingesetzt, um zu vermeiden, dass er zu sehr Fluke-spezifisch wird.)

Fluke benutzt nahezu die gesamte Funktionalität des OSKit (was anhand des Ursprungs von OSKit eher naheliegend ist). Mehr als die Hälfte des Fluke Mikrokernels sind OSKit Funktionen.

4.2 ML/OS

Standard ML ist eine funktionale Programmiersprache (Funktionen höherer Ordnung, statisches polymorphes Typensystem, Exceptions, Continuations). Die Standard ML Implementierung der Universität New Jersey (SML/NJ) wurde so portiert, dass das Laufzeitsystem der Programmiersprache selbst das Betriebssystem eines PCs ist (ML/OS).

SML/NJ ist ein sehr komplexes Unix-basiertes System, das aus etwa 144.000 Zeilen Code verteilt auf über 1.000 Quelltextdateien besteht. Das Ausführungsmodell ist etwas exotisch, da das System komplett ohne Stack funktioniert.

ML/OS wurde über ein Semester am MIT von einem Master-Studenten und einer teilzeitbeschäftigten studentischen Hilfskraft entwickelt, welche beide keine Erfahrung mit OSKit hatten. Die meiste Zeit verbrachten die Studenten damit, sich mit den Details von SML/NJ zu beschäftigen, welches deutlich komplexer war als der OSKit-Code, mit dem es zusammengebracht werden musste.

Literatur

[1] Bryan Ford, Godmar Back, Greg Benson, Jay Lepreau, Albert Lin, and Olin Shivers. The flux OSKit: A substrate for Kernel and language research. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP '97)*, ACM Operating Systems Review, Seiten 38–51, ACM Press, Oktober 1997

[2] Flux Resarch Group, University of Utah, 2006-06-10, www.cs.utah.de/flux