Großer Beleg

zum Thema

Integration des DROPS-Konsolensystems

Mathias Noack
Technische Universität Dresden
Fakultät Informatik

12. Februar 2002

Inhaltsverzeichnis

Kapitel 1

Einleitung

Sehr früh in der Entwicklung der Computersysteme entstanden Lösungen, um die Interaktion mit dem System zu ermöglichen. Besonders wichtig war, einerseits alle Informationen zur Verfügung zu stellen, andererseits diese übersichtlich zu präsentieren. Anfangs wurden die Informationen in Textform verarbeitet und auf dem Bildschirm dargestellt. Eine Möglichkeit zur Präsentation von grafischen Inhalten gab es nicht oder nur rudimentär. Mit wachsender Leistung der Rechnersysteme und Peripheriegeräte wurde diese Art der Informationsdarstellung von grafischen Benutzeroberflächen abgelöst. Der Umgang mit Rechnersystemen gestaltete sich dadurch weitaus einfacher und komfortabler. Dennoch unterstützen heute viele Systeme neben der grafischen Nutzerschnittstelle auch die textbasierte. Im Falle des Betriebssystems Linux stellt sich die textbasierte Schnittstelle als Terminal, die grafische als X-Server dar.

Für das ***D****resden* ***R****eal-Time* ***Op****erating* ***S****ystem* (DROPS) bietet bereits der DROPS-Konsolenserver eine grafische Schnittstelle an. In diesem Beleg soll nun darauf aufbauend eine textbasierte Schnittstelle entwickelt und implementiert werden. Als Motivation diente die Aussicht auf ein interaktives Arbeiten mit L^{4}Linux bzw. DROPS-Anwendungen im Textmodus unter DROPS.

1.1 Aufbau der Arbeit

Der Beleg ist in sechs Kapitel gegliedert und soll den Entwicklungsprozeß von Bestandsaufnahme über Entwurf und Implementierung bis zu einer Leistungsanalyse darstellen.

Das folgende Kapitel zeigt den Stand der Entwicklung zu Beginn dieser Arbeit. Es werden Grundlagen vermittelt, die das Verständnis der weiteren Arbeit erleichtern sollen.

In Kapitel 3 werden verschiedene Modelle zur Integration des DROPS-Konsolensystems in L^{4}Linux sowie in DROPS-Anwendungen diskutiert. Anhand der erforderlichen Grundlagen und Eigenschaften wird eine Auswahl getroffen und diese vorgestellt.

Einige interessante Gesichtspunkte der Implementierung werden in Kapitel 4 herausgegriffen.

In Kapitel 5 werden verschiedene Messungen zur Leistungsbewertung vorgenommen und analysiert.

Zum Schluss zeigt Kapitel 6 zusammenfassende Bemerkungen und liefert Ansätze für weitere Entwicklungen.

Kapitel 2

Stand der Technik

2.1 L⁴Linux

Die Professur Betriebssysteme an der TU Dresden beschäftigt sich im Rahmen des Drops-Projektes (***D**resden **R**eal-Time **Op**erating **S**ystem*) mit einer Portierung von Linux auf den L4-Mikrokern [1]. L4 gehört zu der zweiten Generation von Mikrokernen, die weitaus effizienter und kompakter sind als ihre Vorgänger der ersten Generation. Mittlerweile gibt es Portierungen des L4-Mikrokerns auf die verschiedensten Architekturen wie z. B. DEC-Alpha, StrongARM und MIPS. Innerhalb des Drops-Projektes wird eine L4-Portierung auf die intelbasierte x86-Architektur verwendet. Dieser Fiasco- oder Drops-Mikrokern ist frei verfügbar und implementiert die Version 2 der L4-API [1].

Das monolithische Betriebsystem Linux ist eine frei erhältliche UNIX-Implementierung und ist auf diversen Hardwareplattformen verfügbar. L⁴Linux ist eine Portierung von Linux auf den L4-Mikrokern.

2.2 Das SLIM-Protokoll

Das Slim-Protokoll (Stateless Lowlevel Interface Machine) wurde von Sun Microsystems als simple, hard- und softwareunabhängige Geräteschnittstelle entwickelt [2]. Diese bietet Vielseitigkeit sowie eine optimierte Lowlevel-API, die auch für Kanäle mit geringem Datendurchsatz, z. B. Netzwerke, eine akzeptable Abstraktion der Grafikhardware darstellt. Die Low-level-API des Protokolls besteht aus folgenden Komandos:

SET setzt den exakten Pixelwert für einen rechteckigen Bereich

FILL füllt ein Rechteck mit dem eingestellten Pixelwert

BITMAP füllt ein Rechteck mit einem zweifarbigen Muster

COPY kopiert einen (rechteckigen) Bereich aus dem Framebuffer an eine andere Position

CSCS (color-space convert and scale) wandelt YUV-kodierte Rechtecke nach RGB mit optionaler bilinearer Skalierung

Das SLIM-Protokoll verbietet explizit das Einblenden des Grafikkartenspeichers in Adressräume anderer Prozesse. Stattdessen wird mit einer virtuellen Variante dieses Speichers gearbeitet. Das vom DROPS-Konsolensystem verwendete Protokoll bietet aufgrund einiger Erweiterungen keine vollständige Kompatibilität zum SLIM-Protokoll und wird somit als Pseudo-SLIM-Protokoll (pSLIM) bezeichnet.

2.3 Das DROPS-Konsolensystem

Das DROPS-Konsolensystem wurde von Christian Helmuth [3] im Rahmen seiner Belegarbeit entwickelt und implementiert. Ziel war es, mit dem Konsolensystem, den DROPS-Komponenten eine standardisierte Schnittstelle zur Ein- und Ausgabe von Daten zur Verfügung zu stellen. Der Nachrichtenmechanismus des DROPS-Mikrokerns (L4-IPC) bildet dabei die Kommunikationsgrundlage.

Die Verbindung zwischen einer DROPS-Komponente und der DROPS-Konsole stellt eine zeitlich eindeutige Abbildung dar, um somit eventuelle Konflikte der verschiedenen Ein- und Ausgabeströme zu verhindern. Um diese Abbildungen jedoch variabel zu gestalten, wurden sogenannte *virtuelle Konsolen* eingeführt, zwischen denen beliebig konsistent umgeschaltet werden kann. Eine virtuelle Konsole stellt hierbei immer einen eindeutigen Konsolenzustand dar. Die Funktionsweise und Aufgabe ähnelt der anderer bekannter Systeme wie z. B. Linux oder Solaris.

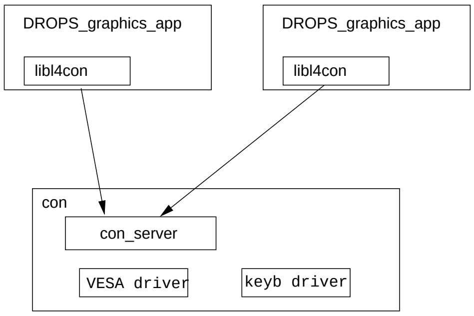

Abbildung 2.1: DROPS-Konsolensystem

Beim Entwurf des DROPS-Konsolensystems entschied sich Helmuth für ein Modell, dass die Schnittstelle zur DROPS-Konsole in einen grafik- und zeichenorientierten Teil trennte. Beide sollten den DROPS-Komponenten in seperaten Servern zur Verfügung stehen. Helmuth konzentrierte sich in seinem Beleg in erster Linie auf Grafikanwendungen. Es enstand eine sehr schlanke, rein grafikorientierte Schnittstelle unter Verwendung des pSLIM-Protokolls und Mechanismen zur Interprozeßkommunikation (IPC) in Form des DROPS-Konsolenservers.

Ein wichtiger Aspekt bei der Betrachtung des DROPS-Konsolensystems ist der *virtuelle Framebuffer*. Er stellt den Grafikkartenspeicher (*Framebuffer*) im Adressraum des DROPS-Konsolenservers dar. Der virtuelle Framebuffer enthält Informationen die benötigt werden, um eine Konsole vollständig rekonstruieren zu können. Zwischen den virtuellen Konsole kann beliebig umgeschalten werden. Jeweils nur eine Konsole befindet sich im Vordergrund d. h. sichtbar auf dem Bildschirm, wobei alle anderen im Hintergrund agieren. Nach dem Zurückschalten einer Konsole in den Vordergrund, wird der virtuelle Framebuffer vollständig in den Framebuffer der Grafikkarte kopiert. Alle Änderungen der Vordergrundkonsole werden nun direkt auf dem Grafikkarten-Framebuffer umgesetzt.

2.4 Das Linux-Konsolensystem

Um das DROPS-Konsolensystem in L^{4}Linux zu integrieren, wird eine genauere Betrachtung des Linuxkerns insbesondere der Linux-Konsole notwendig. Der modulare Aufbau des Linuxkerns, und die hohe Flexibilität erleichtert hierbei nicht nur die Analyse und das Verständnis, sondern auch das spätere Ersetzen von konsolenspezifischen Komponenten. In Abb. 2.2 wird die Struktur der Linux-Konsole gezeigt.

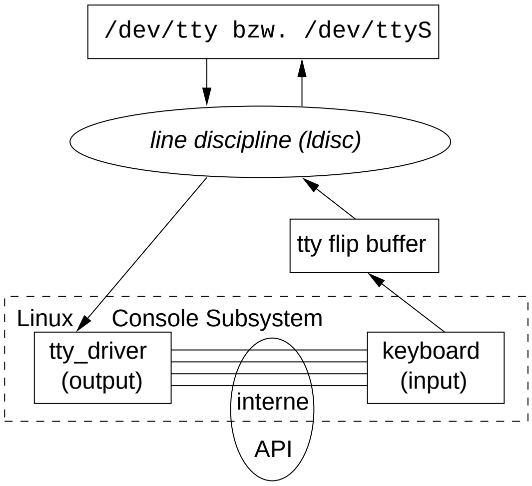

Abbildung 2.2: Linux-Konsolensystem

Eine wichtige Rolle innerhalb des Linux Konsolen Systems spielt das tty[1]-Subsystem [4]. Es kontrolliert die Kommunikation zwischen Anwendung und lokal oder über Netzwerk angeschlossener Geräte. Weiterhin wird der physische Datenfluß sowie der Zugang zu den entsprechenden Geräten vom tty-Subsystem geregelt. Als Ausgabekomponente für Zeichendaten dient der tty-Treiber. Dieser hat direkten Zugriff auf die Hardware (tty-Gerät) oder Pseudohardware (pty-Gerät). Im Falle der Linux-Konsole stellt sich der tty-Treiber als Konsolentreiber dar.

[1]tty steht für *Teletype*

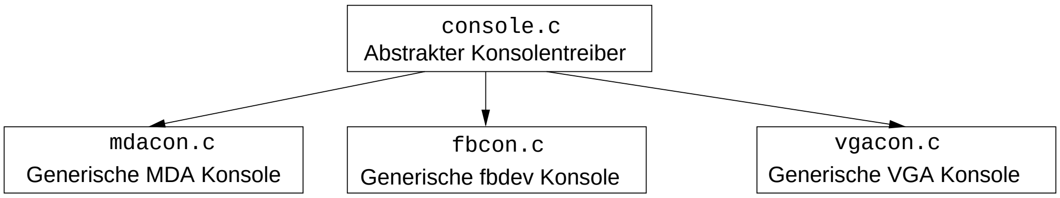

Abbildung 2.3: Linux Konsolentreiber (tty-Treiber)

Konsolentreiber älterer Linux-Versionen waren eng an die VGA Hardware und dem VGA kompatiblen Textmodus gebunden. Um jedoch auch andere Arten von Konsolen zu unterstützen wurde eine schmale Abstraktionsschicht eingeführt, der sogenannte abstrakte Konsolentreiber (Abb. 2.3). Die generische Konsole, die letztendlich die Kontrolle über die Ausgabefunktionen übernimmt, wird je nach Konfiguration unterschiedlich abgebildet. Dabei spielt es keine Rolle, ob es sich um eine Text- (VGA, MDA) oder Framebuffer-Konsole (`fbcon.c`) handelt. Folgende Grundfunktionen sind Bestandteil der generischen Schnittstelle, welche der abstrakten Konsole zur Verfügung stehen müssen:

putc stellt ein Zeichen dar

putcs stellt eine Zeichenkette dar

scroll verschiebt einen Bereich des Bildschirms

clear löscht einen Block von Zeichen

bmove verschiebt einen Block von Zeichen

cursor setzt oder löscht den Cursor

Die Trennung des Konsolentreibers in einen abstrakten sowie generischen Teil fördert die Modularität und Flexibilität des gesamten Linux-Konsolensystems.

2.4.1 Framebuffer Device Subsystem

Das *Framebuffer Device Subsystem* wurde von Helmuth als Ansatzpunkt für eine Integration des Drops-Konsolensystem vorgeschlagen und wird deshalb näher betrachtet. Seit einigen Jahren beschäftigt sich das *Linux Framebuffer Device Project* [5] mit der Entwicklung dieses Subsystems, mit dem Ziel Anwendungen eine standardisierte Abstraktion für framebufferbasierte Ausgabegeräte zu bieten. Damit besteht die Möglichkeit, über eine wohl definierte Schnittstelle auf Videogeräte zuzugreifen, ohne deren spezifischen Aufbau zu kennen.

Ein Framebuffer stellt sich als linearer Speicherbereich mit festgelegter Kodierung der Pixelwerte in Abhängigkeit vom Darstellungsmodus dar. Aus diesem Puffer sendet die Grafikhardware die nötigen Informationen jedes Bildes an das Ausgabemedium. Diese Bildinformationen werden auch *Frames* genannt. Zu den Aufgaben des Framebuffer Device Subsystems

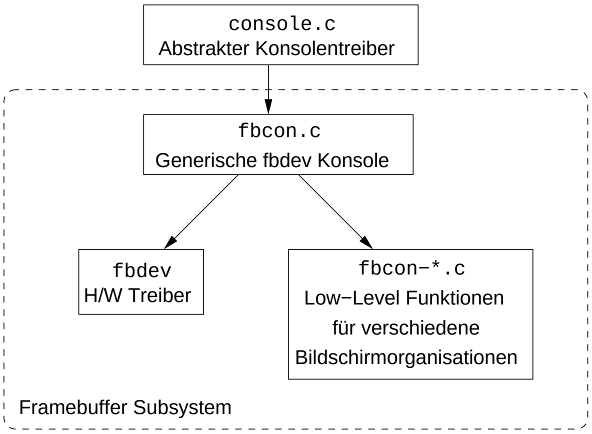

Abbildung 2.4: Framebuffer Subsystem

zählt, den Grafikkarten-Framebuffer in den linearen Adressraum von Linux einzublenden. Die Framebuffer-Konsole (`fbcon`) bildet hierbei den generischen Teil der Linuxkonsole, wobei das Framebuffer-Gerät (`fbdev`) den eigentlichen Gerätetreiber darstellt. Der Zugriff auf dieses Gerät erfolgt über einen speziellen Geräteknoten (`/dev/fbx`). Neben Standardoperationen wie Öffnen, Schließen, Lesen und Schreiben können Anwendungen sich über die standardisierte Geräteschnittstelle des Framebuffer-Gerätes den Framebuffer der Grafikkarte in ihren Adressraum einblenden lassen (*mmap*).

Es gibt mit dem GGI-Projekt [6] und dem KGI-Projekt [7] noch weitere Versuche den Linuxkern um eine grafische Schnittstelle zu erweitern. Sie haben allerdings für diesen Beleg keine Bedeutung und werden deshalb nicht weiter betrachtet.

2.5 Terminals

Ein Terminal ist eine sehr einfach gehaltene Ein- und Ausgabekomponente, bestehend aus einem Bilschirm und einer Tastatur. Die Kommunikation zu einem Rechner findet meist über eine serielle Verbindung bzw. über eine Netzwerkverbindung statt. Zeichendaten und Kontrollsequenzen werden effizent über diese Verbindung zum Terminal übertragen und dort interpretiert. Es gibt verschiedene Typen von Terminals, z. B. VT100 oder xTerm. Sie unterscheiden sich in der Anzahl und Bedeutung der Kontroll- und Escapesequenzen, mit denen Bildschirmoperationen wie Cursor setzen oder Zeilenumbruch kodiert werden. Eine Standardisierung der Sequenzen fand bisher noch nicht statt.

Ein Terminal zu emulieren bedeutet, den Rechner dazu zu bewegen, sich wie ein Terminal zu verhalten. Die Linux-Konsole emuliert den *Linux*-Terminaltyp.

2.6 DSI

Das DROPS *Streaming Interface* (DSI) [8] wurde im Rahmen des DROPS-Projektes entwickelt und bietet ein packetorientiertes *zero-copy*-Protokoll für die Kommunikation über Adressraumgrenzen hinaus an. Zwei Shared-Memory-Bereiche (Kontroll- und Datenbereich) dienen als Transportmechanismus und bilden zusammen mit einer Bibliothek von Servicefunktionen die Datenschnittstelle der Komponenten. Der Datenaustausch findet im Datenbereich statt. Dabei werden Seiten des virtuellen Adressraumes des Senders mit Hilfe des DROPS-Mikrokerns in den des Empfängers eingeblendet. Es obliegt den beteiligten Komponenten, diesen Bereich zu strukturieren. Eigentlich übertragen werden nur Referenzen auf bestimmte Bereiche des gemeinsamen Adressraumes. Der Kontrollbereich hält diesbezüglich DSI-interne Deskriptoren für Daten und Pakete bereit. Der indirekte Zugang zu den Übertragungsdaten mittels Referenzen, die in einem Ringpuffer organisiert werden, bietet eine flexible Nutzung des DSI als klassisches Erzeuger-Verbraucher-System. Auf Senderseite werden Pakete in einen Datenstrom gegeben, die der Empfänger auf der anderen Seite entnimmt. Dieser Strom von Datenpaketen ist dabei unidirektional. Mittels DSI können Leistungsschwächen, wie z. B. doppeltes Kopieren mit möglichen Inkonsistenzen herkömmlicher Nachrichtenmechanismen, beseitigt werden.

Kapitel 3

Entwurf

Die Zielstellung dieser Arbeit ist die Integration des DROPS-Konsolensystems in bestimmten Anwendungsszenarien wie z. B. L4Linux oder DROPS-Anwendungen. Es gibt zwei Ansätze, die sich in der Praxis durchgesetzt haben. Zum einen soll den Anwendungen eine grafisch orientierte Nutzerschnittstelle zur Verfügung gestellt werden. Mit dem DROPS-Konsolensystem wurde eine solche bereits entwickelt und implementiert. Allerdings können bisher nur DROPS-Anwendungen darauf zugreifen.

Im Falle von L4Linux müsste der X-Server als grafische Benutzeroberfläche an das DROPS-Konsolenprotokoll angepaßt werden. Aufgrund des umfangreichen X-Serverquelltextes ist eine längere Analyse- und Einarbeitungszeit nötig. Zudem dürfte die mangelhafte Dokumentation des Quelltextes eine Integration des DROPS-Konsolensystems erschweren. Die Entwicklung einer geeigneten Schnittstelle für Grafikdaten soll somit vorerst in den Hintergrund treten und wird in diesem Beleg nicht weiter betrachtet.

Als Alternative bietet sich die Entwicklung einer rein textbasierten Nutzerschnittstelle an. Diese sollte ein erstes Arbeiten mit L4Linux unter DROPS ermöglichen und würde den Umfang dieses Beleges nicht übermässig ausdehnen. Diese Arbeit wird sich deshalb im weiteren Verlauf darauf konzentrieren, L4Linux sowie DROPS-Anwendungen die Möglichkeit zu bieten, Text auf den DROPS-Konsolen darzustellen.

Auf Grund der unterschiedlichen Voraussetzungen und Anforderungen an eine Textausgabe ist eine getrennte Betrachtung der beiden Anwendungen nötig. Zum einem soll ein L4Linux-Stub entwickelt werden, der den Zugriff auf die Hardware unterbindet und stattdessen die DROPS-Konsolenschnittstelle nutzt. Zum anderen gilt es, den DROPS-Anwendungen eine transparente Schnittstelle für zeichenorientierte Ausgaben zur Verfügung zu stellen.

3.1 Integration mit DROPS-Anwendung

3.1.1 Der virtuelle Framebuffer

In [3] wurde insbesondere auf die große Datenmenge zur Bilddarstellung der einzelnen DROPS-Konsolen und die daraus folgenden Probleme eingegangen. Direkt beeinflusst durch diese

Datenmenge werden vor allem der Pufferspeicher für IPC-Nachrichten und der virtuelle Framebuffer jeder Konsole.

- sparsame Beispielkonfiguration: 640x480 Bildpunkte bei 16 Bit Farbtiefe (2^{16} Farben)

$$640 \cdot 480 \cdot 2 = 600\,\text{kbyte}$$

- übliche Konfiguration aktueller Desktopsysteme: 1024x768 bei 24 Bit (2^{24} Farben)

$$1024 \cdot 768 \cdot 3 = 2,25\,\text{Mb}$$

Bei entsprechend vielen Konsolen lässt sich der enorme Speicherbedarf nicht mehr rechtfertigen. Als Lösung wurde von Helmuth vorgeschlagen, die Maximalgröße des Empfangspuffers für IPC-Nachrichten beim Öffnen mit anzugeben.

Im Falle von reinen Textanwendungen ist das Führen eines virtuellen Framebuffers allerdings unnötig. Es würde genügen, statt der Bitmapdaten jedes Zeichens, das Zeichen selber in einem Textpuffer zu halten. Das Resultat wäre eine erhebliche Reduzierung der Speicherkosten (siehe Tab. 3.1).

Schriftmaße	Bitmapgröße	Zeichengröße	mit Attributen
10x6	60 bit	8 bit	16 bit
12x8	96 bit	8 bit	16 bit
16x10	160 bit	8 bit	16 bit

Tabelle 3.1: Speicherkosten unterschiedlicher Schriftarten

Da das pSlim-Protokoll keine eigene Funktionalität zur Textausgabe besitzt, stehen dem Konsolensystem nur Grafikdaten zur Verfügung. Somit ist das Führen eines Textpuffers anstelle des virtuellen Framebuffers in der Konsole vorerst nicht möglich. Überhaupt sollte das Verwalten solcher Puffer, Aufgabe der Anwendung sein, wie es z. B. bei Linux der Fall ist. Das bietet folgende Vorteile:

- keine Kommunikation bei Änderungen wenn Anwendung sich nicht im Vordergrund befindet
- keine Managmentfunktionalitäten bzgl. eines Textpuffers im Konsolensystem
- Erhöhung der Zuverlässigkeit
- Senkung des Speicherbedarfs auf Seiten des Konsolensystems

Um Grafikanwendungen die Möglichkeit zu bieten, die Ausgabefunktionen der Konsole zu nutzen, ohne den eigenen virtuellen Framebuffer verwalten zu müssen, soll dieser selektiv vom Drops-Konsolensystem geführt werden.

3.1.2 Ansätze

In engem Zusammenhang mit einer Textausgabe stehen oft Aufgaben wie das *Textrendering* und die Verwaltung eines Textpuffers. Im Folgenden werden nun vier verschiedene Modelle vorgestellt, die die unterschiedlichen Möglichkeiten der Verteilung dieser Aufgaben erläutern.

Modell 1: Ein seperater Textserver Dieses Modell wurde von Helmuth favorisiert und auch schon im Rahmen eines Komplexpraktikums [9] implementiert.

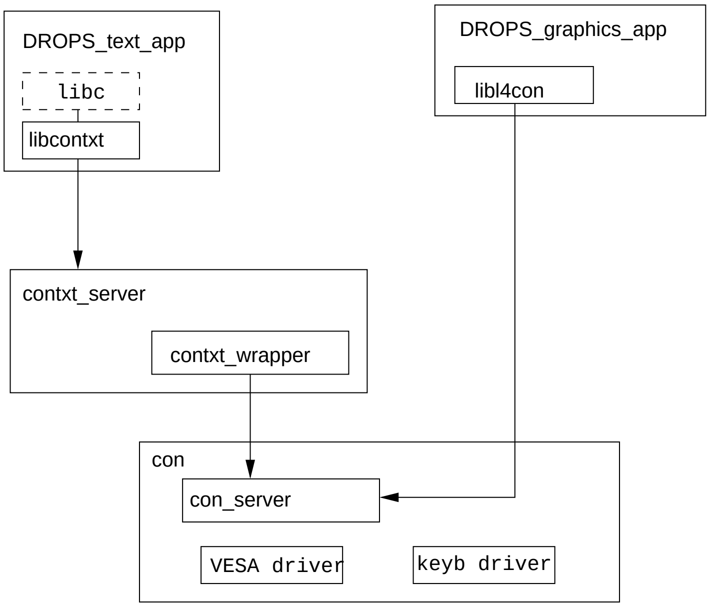

Abbildung 3.1: Seperater Textserver

Die Textausgabefunktionalität wird hier in einen seperaten Textserver gekapselt. Dabei übernimmt der Server die Verwaltung des Textpuffers sowie die Umschaltung zwischen den virtuellen Textkonsolen, wobei jede Textanwendung ihre eigene Textkonsole anfordern kann. Es besteht die Möglichkeit, alle Textkonsolen auf eine Drops-Konsole *multiplexen* zu lassen oder jeder einzelnen eine eigene zuzuweisen. Ein seperater *Textwrapper-Thread* übernimmt die Aufgabe des Textrenderings und gibt die Bitmapdaten jedes Zeichens an die Drops-Konsole weiter. Der Nachteil dieses Modells ist, dass eine zusätzliche Komponente an der Kommunikation zwischen Anwendung und Drops-Konsolensystem benötigt wird. Das führt nicht nur zu höheren Kommunikationskosten, sondern stellt auch eine zusätzliche Fehlerquelle dar, was die vorangegangene Arbeit an diesem Modell bestätigte.

Modell 2: Einblenden des Framebuffers Das Einblenden des Framebuffers der Grafikkarte in den linearen Adressraum der Anwendung verringert erheblich die Kommunikation zur Drops-Konsole. Es findet ein sogenanntes *zero-copy* statt, also ein direktes Schreiben auf den Framebuffer ohne zusätzliche Kopieroperationen und Kommunikationskosten. Heutzutage bieten viele Grafikkarten beschleunigte Operationen an, die direkt auf dem Framebufferspeicher der Karte arbeiten (z. B. kopieren, füllen, clipping). Diese Hardwareunterstützung kann auch bei eingeblendeten Framebuffern genutzt werden, um Standardfunktionen zu optimieren. Das pSLIM-Protokoll wird allerdings vollständig umgangen. Dadurch ist das Modell weniger universell einsetzbar. Weiterhin lässt sich das sogenannte Clipping mit eingeblendetem Speicher nicht realisieren. Grund hierfür ist, dass sich nur Seiten architekturbedingter Größe in den Adressraum der Anwendung einblenden lassen. Die Granularität Speicherseite ist jedoch zu grob, um ein Clipping zu realisieren.

Modell 3: contxt-Bibliothek ohne pSlim-Erweiterung Eine Bibliothek übernimmt transparent für die Anwendung das Umwandeln der Zeichendaten in Bitmapdaten. Somit müssen Schriftattribute, wie Höhe und Breite einer Schrift, nicht mehr dem Konsolensystem bekannt sein. Ebenfalls wird die Organisation des Textpuffers und die Bereitstellung einer einfachen zeichenorientierten Schnittstelle für Textausgaben der Anwendung in dieser Bibliothek realisiert.

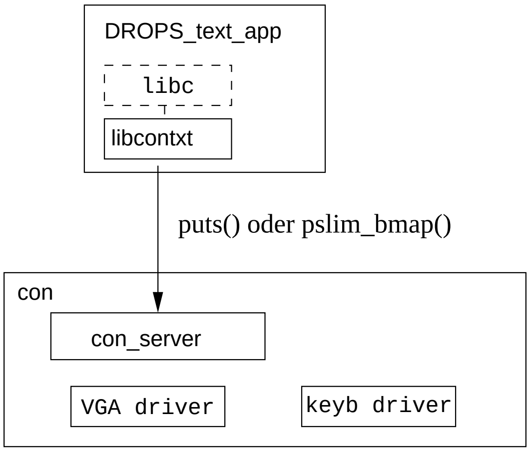

Abbildung 3.2: contxt-Bibliothek mit oder ohne Erweiterung

Das pSlim-Protokoll und die Konsolenfunktionalität bleiben hierbei unverändert. Nachteilig wirkt sich allerdings die große Datenmenge aus, die zu übertragen ist. Es handelt sich dabei um die fertig gerenderten Bitmapdaten eines Zeichens oder einer Zeichenkette (siehe Tabelle 3.1, Seite 14). Mit dem DROPS Streaming Interface (DSI) könnte man einen Kommunikationsmechanismus nutzen, der über einen gemeinsamen Speicherbereich den Austausch von großen Datenmengen sehr schnell ermöglicht. Es übernimmt auch die vollständige Synchronisation zwischen den Kommunikationspartnern und organisiert die Paketwarteschlange. Bei einer Übertragung von ganzen Zeichenketten oder gar einzelnen Zeichen könnte jedoch der Kommunikationsaufwand zur Synchronisation der DSI-Komponenten größer sein, als das

Senden einer entsprechenden IPC-Nachricht. DSI lässt sich auch über Rechnergrenzen hinaus nicht verwenden, da es momentan noch keine Komponente gibt, die das Einblenden von Adressräumen auf ein Netzwerkprotokoll abbildet.

Modell 4: contxt-Bibliothek mit pSlim-Erweiterung Das Ergänzen des pSlim-Protokolls um die Funktionalität der Textausgabe bedeutet, dass das Drops-Konsolesystem keine rein grafikorientierte Komponente mehr darstellt. Erweiterungen einer *trusted component*, wie der des Konsolensystems, stehen im engen Zusammenhang mit Fragen der Zuverlässigkeit. Im Allgemeinen gilt: Je geringer die Funktionalität, desto mehr Sicherheit kann geboten werden. Dieses Prinzip wird vor allem bei mikrokernbasierten Betriebssystemen verfolgt. Um ein fehlerfreies Arbeiten des Konsolensystem zu gewährleisten, sollten das Hinzufügen zusätzlicher Funktionalität genau abgewogen werden. Dennoch besitzt die pSlim-Erweiterung auch entscheidende Vorteile. Zum einen können nun reine Zeichendaten übertragen und auf der Konsolenseite verarbeitet werden. Die Möglichkeit zur Kommunikation über Rechnergrenzen hinaus bleibt bestehen und wird mit der Reduzierung der zu sendenen Daten noch begünstigt. Das Führen eines Textpuffers in der Drops-Konsole ist aufgrund der zugänglichen Zeichendaten denkbar. Dennoch sollte davon abgesehen werden, da dies den Funktionsumfang und somit die Fehleranfälligkeit erhöhen würde.

3.1.3 Entwurfsentscheidung

Bei der Betrachtung der Vor- und Nachteile der verschiedenen Ansätze, scheint die Entwicklung einer Bibliothek für Drops-Anwendungen aus einer Kombination der Modelle 3 und 4, die flexibelste Lösung zu sein. Zum einen soll der Textpuffer jeder Konsole vollständig in der Bibliothek geführt, zum anderen das pSlim-Protokoll um eine Textausgabemöglichkeit erweitert werden. Daraus ergeben sich folgende Vorteile:

- die zu übertragene Datenmenge bleibt gering
- Organisation des Textpuffers auf Anwendungsseite
- nur geringe Erweiterung des Konsolensystems um ein Textschnittstelle
- Minimierung des Kommunikationsaufwandes
- universell und flexibel einsetzbar
- über Rechnergrenzen hinaus verwendbar

Das Übertragungsprotokoll soll selektiv um die Möglichkeit der Verwendung von DSI erweitert werden. Dies ist nicht nur hinsichtlich späterer Leistungsmessungen interressant.

3.2 Integration in L^4Linux

Ziel dieses Entwurfs sollte die Virtualisierung der Ein- und Ausgabefunktionen von L^4Linux sein. Dabei gilt es, den direkten Zugriff auf die Hardware zu unterbinden und stattdessen das pSLIM-Protokoll als Schnittstelle zum DROPS-Konsolensystem zu verwenden. Sämtliche L^4Linux-Konsolen können auf eine DROPS-Konsole abgebildet werden. Die Frage, welche Komponente die Organisation des Textpuffers übernimmt, stellt sich bei L^4Linux ebenso wenig, wie die des Textrenderings. Dort übernehmen die grafischen Ausgabekomponenten auf der L^4Linux-Seite diese Aufgaben selbst. In Abb.3.3 werden die zwei Integrationsansätze des DROPS-Konsolesystems in L^4Linux aufgezeigt. Im Folgenden werden diese analysiert.

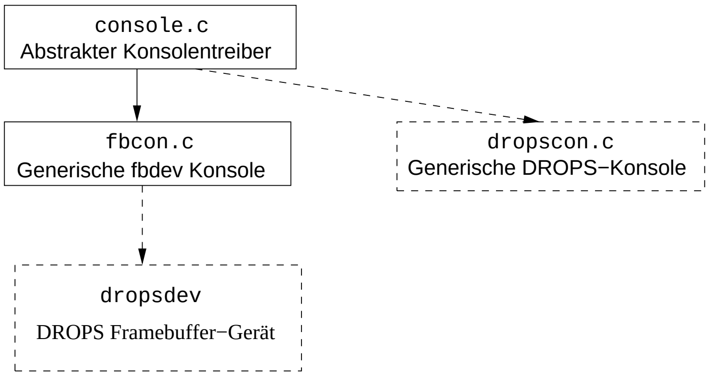

Abbildung 3.3: Möglichkeiten der Integration

3.2.1 L^4Linux-Stub als Framebuffer-Gerät

Die vollständige Kontrolle über die Ausgabefunktionen übernimmt bei einem Framebuffer-Linux das Framebuffer Subsystem. Innerhalb des Subsystems kann der L^4Linux-Stub als eigenständiges Framebuffer-Gerät (`fbdev`) entwickelt werden, wobei die Framebuffertreiber-Schnittstelle auf das pSLIM-Protokoll abgebildet wird. Dieser Integrationsschritt wurde bereits in [3] kurz beschrieben. Framebuffer-Geräte besitzen unter anderem die Möglichkeit, den Framebuffer der Grafikkarte in den linearen Adressraum der Anwendung einzublenden. Modell 2 des vorhergehenden Abschnittes beschreibt die Vor- und Nachteile des Einblendens. Der hierarchische Aufbau des Framebuffer-Subsystems verhindert den direkten Zugriff der Framebuffer-Geräte auf den Textpuffer der jeweiligen Konsole. Dieser Zugriff ist jedoch notwendig für die Rekonstruktion des Bildschirms nach dem Umschalten der entsprechenden DROPS-Konsole in den Vordergrund. Im Falle von Linux-Konsolen übernimmt diese Aufgabe das Linux-Konsolensystem selbstständig. Aufgrund der oben genannten Eigenschaften bietet diese Variante nicht die idealen Voraussetzungen für einen Integrationsansatz.

3.2.2 L4Linux-Stub als generische Konsole

Als weitere Möglichkeit bietet sich die Entwicklung des Linux-Stubs als eigenständige generische Konsole an. Der standardisierte strukturelle Aufbau der Konsole erleichtert die Einbettung des pSLIM-Protokolls. Das Einblenden des Grafikkartenspeichers kann vermieden werden und der Zugriff auf den Textpuffer ist gewährleistet. Somit soll im Rahmen dieses Beleges das DROPS-Konsolensystem mittels eines L4Linux-Stubs als generische Konsole in L4Linux integriert werden.

3.2.3 Terminal Emulation

Unter Linux lassen sich Text-Terminals emulieren, die das Darstellen von Text sowie die Organisation des Bildschirms übernehmen. Die dafür benötigten Informationen müssen in Form von Zeichendaten und Kontroll- bzw. Escapesequenzen bereitgestellt werden.

Das DROPS-Konsolensystem selbst könnte um Terminalfunktionalitäten erweitert werden, so dass nicht nur eine einfache Textausgabe erfolgt, sondern auch eine Interpretation der übermittelten Kontrollinformationen. Diese Erweiterung ist von weitaus größerem Umfang als ein einfaches Hinzufügen einer Textausgabeschnittstelle. Wie bereits in Kapitel 3.1.2 (Modell 4, Seite 16) erwähnt, ist eine derartige Ausweitung des Funktionsumfanges nicht anstrebenswert.

Innerhalb von Linux übernimmt die Emulation eines Terminals zum größten Teil der Linux-Konsolentreiber (*tty-Treiber*). Es handelt sich hierbei um den Terminaltyp *Linux*. Dort werden Zeichendaten und Kontrollsequenzen interpretiert und auf entsprechende Hardwarebefehle abgebildet. Die entsprechenden Hardwarebefehle müssten durch das pSLIM-Protokoll ersetzt werden. Es wird somit in die abstrakte Schicht des Konsolensystems eingegriffen (siehe Abb. 2.3, Seite 10). Diese Schicht im Linux Konsolensystem soll hardwareunabhängig sein und Flexibilität bieten. Mit Anpassungen an das pSLIM-Protokoll werden diese Eigenschaften eingeschränkt. Da das Linux Konsolensystem fest im Linuxkern verankert ist, wird somit der Kern selber verändert. Deshalb sollte eine modulare Lösung ohne Anpassung der Linuxquellen gefunden werden.

Kapitel 4

Implementierung

4.1 Änderungen am DROPS-Konsolensystem

Um das Konsolensystem performant und sicher zu gestalten, sollten reine Grafikanwendungen den virtuellen Framebuffer möglichst selbst in ihrem Adressraum verwalten. Alternativ dazu kann diese Aufgabe aber auch von der DROPS-Konsole übernommen werden. Beim Öffnen einer DROPS-Konsole wird ein entsprechender Parameter verwendet (`CON_VFB` oder `CON_NOVFB`), der den gewünschten Modus angibt. Textanwendungen führen über eine transparente Bibliothek ihren Textpuffer selbst und benötigen daher keinen virtuellen Framebuffer. Weiterhin stehen den Anwendungen diverse Bildschirmauflösungen zur Verfügung (800x600, 1024x768). Mit der putcs-Funktion (siehe Datei `vc.c`) wurde das pSLIM-Protokoll um eine Funktionalität erweitert, die eine Textausgabe über die Konsole ermöglicht. Für jedes Zeichen des auszugebenden Textes werden die jeweiligen Bitmapdaten bestimmt und mittels der Standard-pSLIM-Funktion `bmap` in den Framebuffer der Grafikkarte übertragen.

4.2 Die contxt-Bibliothek

Mit der contxt-Bibliothek (`libcontxt`) wird nicht nur die Komplexität des Textpuffermanagments vor der DROPS-Anwendung verborgen, sondern es wird auch gleichzeitig eine gewohnte Schnittstelle zur Aus- und Eingabe von Textdaten zur Verfügung gestellt. Jeder Anwendung wird ein Textpuffer (`vtc_scrbuf`) zugewiesen, der standardmäßig die Größe eines kompletten Textbildschirms hat. Darüber hinaus kann optional der Textbuffer beim Initialisieren der Bibliothek erweitert werden. Der so entstandene sogenannte *history buffer* verhindert, dass Text außerhalb des sichtbaren Bereiches des Bildschirms verloren geht. Ein Fenster zeigt dabei immer auf den aktuellen Bereich im Textbuffer. Mittels *scrolling* lässt sich dieses Fenster verschieben.

Daraus ergibt sich das Problem des kompletten Neuzeichnens —redraw— des Bildschirms. Dies bedeutet, dass der aktuell sichtbare Bereich des Textbuffers neu dargestellt werden muss. Ein vollständiges Neuzeichnen tritt sehr häufig auf und beeinflusst somit die Gesamtperformance der Ausgabe erheblich. Aus diesem Grund wurden zwei Varianten implementiert, die ein Neuzeichnen realisieren.

Zum einen gibt es die Möglichkeit, innerhalb des Framebuffers den entsprechenden Bereich mittels des pSLIM-Befehls `copy` direkt zu kopieren. Besonders geeignet ist dieses Verfahren für Grafikkarten, die eine beschleunigte Kopieroperation anbieten. Für die momentan verwendete Karte existiert allerdings keinerlei Hardwareunterstützung und somit ist das Kopieren innerhalb des Framebuffers sehr teuer (vgl. Tabelle 5.3 und 5.4, Seite 27).

Zum anderen kann jede Zeile des sichtbaren Bereiches aus dem Textpuffer neu gezeichnet werden. Hierbei fallen die Kommunikationskosten mehr ins Gewicht, da pro Bildschirmzeile ein IPC-Aufruf nötig ist. Beide Varianten stehen der Anwendung zur Verfügung, wobei standardmäßig das zeilenweise Neuzeichnen verwendet wird. Das Kopieren innerhalb des Framebuffers lässt sich über ein Makro (`CONTXT_COPY`) zur Compilierzeit auswählen. Die von der contxt-Bibliothek implementierte Schnittstelle stellt sich wie folgt dar:

printf gibt eine Zeichenkette aus

putchar gibt ein Zeichen aus

getchar liest ein Zeichen

readln liest eine Zeichenkette ein (mit Zeilenumbruch)

contxt_write Low-Level-Ausgabe eines Zeichens oder einer Zeichenkette

contxt_read Low-Level-Eingabe eines Zeichens

Bereits bekannte und standardisierte Funktionsnamen wurden verwendet, um dem Nutzer eine gewohnte Programmierumgebung zu bieten. Im Falle der printf-Funktion wurde die komplette Funktionalität beibehalten. Änderungen fanden nur am ausgabespezifischen Teil statt. In Anlehnung an eine noch zu entwickelnde DROPS-C-Bibliothek kann jede contxt-Funktion auf eine einfache read/write-Schnittstelle abgebildet werden. Mit `contxt_read()` und `contxt_write()` ist eine solche schmale Schnittstelle implementiert worden. Die jetzigen Aufgaben der contxt-Bibliothek, wie Formatieren der Zeichenketten und Textpuffermanagment könnten somit in einer eigenen C-Bibliothek gekapselt werden. Damit reduzieren sich der Aufgabebereich der contxt-Bibliothek auf die grundlegendste Ein- und Ausgabefunktionen.

Als Übertragungsprotokoll steht neben pSLIM auch noch DSI zur Verfügung. Mittels `USE_DSI` (`contxt_dsi.h`) lässt sich DSI verwenden. Das DSI-Interface wurde so implementiert, dass Anwendung und Sender den selben Thread darstellen (siehe Abb. 4.1, Seite 23).

4.3 L^{4}Linux-Stub

Generische Konsolen müssen festgelegte Initialisierungs- und High-Level-Funktionen bereitstellen, die in der `consw`-Struktur (`console.h`) abgelegt sind und von der abstrakten Konsole aufgerufen werden. An dieser Stelle wird der gerätespezifische Teil der Ausgabe und die

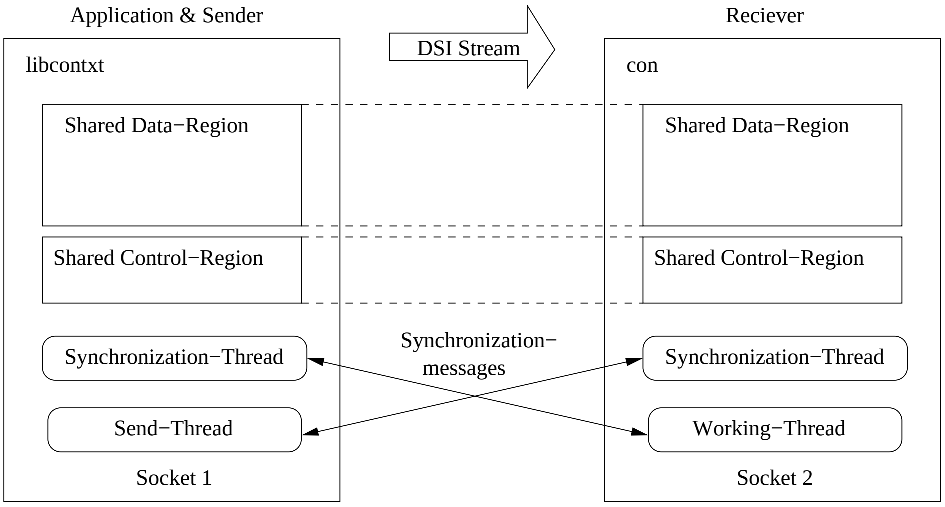

Abbildung 4.1: Implementiertes DSI-Modell

Kommunikation zum Drops-Konsolensystem über das pSlim-Protokoll realisiert. Während des Initialisierungsvorganges stellt der L4Linux-Stub eine Anfrage zum Öffnen einer Drops-Konsole. Dieser dient als weiterer Kommunikationpartner und repräsentiert alle Linux-Konsolen innerhalb des Drops-Konsolensystems. Die Aufgaben des L4Linux-Stubs verteilen sich auf drei Komponenten. Die erste Komponente stellt die eigentliche generische Konsole (`main.c`) dar und implementiert die oben erwähnte Schnittstelle zur abstrakten Konsole. Weiterhin gibt es einen Eingabe- und einen Ausgabeteil.

4.3.1 Eventhandler

Der sogenannte Eventhandler (`evh.c`) ist ein L4-Thread und für die Weiterleitung der ankommenden Drops-Konsolenereignisse an L4Linux verantwortlich. Momentan wird zwischen zwei Ereignistypen unterschieden.

`EV_KEY` - Tastaturereignisse

`EV_CON` - konsolenspezifische Ereignisse

Die Ereignisstruktur ist jedoch so implementiert, dass sich neue Typen ohne weiteres hinzufügen lassen. Besonders interessant ist das konsolenspezifische redraw-Ereigniss (`EV_CON_REDRAW`). Es wird benötigt, um dem L4Linux-Stub mitzuteilen, dass er wieder als Vordergrundkonsole bezüglich des Drops-Konsolensystems agiert. Dies verlangt ein komplettes Neuzeichnen des sichtbaren Bereiches des Textbuffers der aktuellen Linux-Konsole (`dropscon_redraw()` in `main.c`). Zwischen Drops-Konsolen wird laut Vereinbarung mittels Alt_Gr-Fx, zwischen Linux-Konsolen bekannterweise mittels Alt-Fx umgeschaltet.

4.3.2 Commandhandler

L^4Linux-Anwendungen sollten möglichst konfliktfrei auf das Linux-Konsolensystem zugreifen können. Aus diesem Grund sind die generischen Schnittstellenfunktionen nicht direkt auf das pSLIM-Protokoll abzubilden. Der für das pSLIM-Protokoll verwendete IPC-Nachrichtenmechanismus ist eine synchrone Kommunikationsform. Auf Grund dessen muss der Sender, in diesem Fall der L^4Linux-Stub, nach dem Senden einer Nachricht auf eine Antwort warten. In L^4Linux gibt es eine bestimmte Prioritätenreihenfolge, die in jedem Falle eingehalten werden muss.

- Linux-Hauptserver (niedrigste Priorität)
- Bottom-Half-Thread
- Top-Half-Thread (höchste Priorität)

Wenn ein Bottom-Half-Thread eine Ausgabe machen will, muss er als Sender warten (blockierend) und ist verdrängbar. Falls der Linux-Hauptserver dann an der Reihe ist, führt das zu einem "*Scheduling in interrupt*"-Fehler. Dieser tritt immer dann auf, wenn der Hauptserver-Thread feststellt, dass ein Scheduling stattfand, bevor eine Interruptroutine abgschlossen wurde.

Zur Lösung dieses Problems bietet sich ein klassisches Erzeuger-Verbraucher-System an. Die generische Konsole trägt während eines Funktionsaufrufes das Funktionskommando und die benötigten Parameter in eine Komandoliste (`comh_list`) ein. Danach kehrt die Funktion wieder zurück, blockiert also nicht. Der sogenannte Commandhandler (`comh.c`) ist ein L^4Linux-Kernelthread und verarbeitet jedes in der Liste eingetragene Kommando nacheinander ab. Die Kommandos werden in pSLIM-Protokoll-Befehle übertragen und an die DROPS-Konsole weitergegeben. Das Aufrufen einer Funktion und das Ausführen werden voneinander gekapselt. Die Kommunikation zwischen dem L^4Linux-Stub und der DROPS-Konsole ist daher asynchron.

Ein Problem ist allerdings das Überlaufen der Kommandoliste. Ein *Flush-Flag* (`flush_comh_requests`) signalisiert diesen Zustand. Alle folgenden Kommandos werden nur noch auf dem Textpuffer ausgeführt und gehen somit nicht verloren. Der Commandhandler leert bei gesetztem Flush-Flag die Liste und führt anschließend ein komplettes Neuzeichnen des Textpuffers aus.

Kapitel 5

Leistungsbewertung

Für die Leistungsbewertungen der implementierten Komponenten spielen vor allem die Ausführungszeiten der einzelnen Ausgabefunktionen eine Rolle. Besonders interessant ist die benötigte Dauer für die Ausgabe eines Zeichens, einer Bildschirmzeile, sowie des ganzen Bildschirmes mit unterschiedlicher Farbtiefe. Als Hardwareplattform diente ein Intel Pentium 90 MHz mit 64 Mb RAM und eine ATI Rage XL Grafikkarte mit 8 Mb DRAM. Der für die Framebuffer-Testreihe benötigte Referenzrechner war ein Pentium 4, 1,6 GHz mit einer Matrox Millennium G550 mit 32 Mb DDR-RAM. Unter Verwendung des *Time-Stamp Counters* (TSC) des Pentiumprozessors wurden die Messwerte im Testprogramm ermittelt. Als Beispielkonfiguration diente eine Auflösung von 640x480 bei verschiedenen Farbtiefen (16 Bit, 24 Bit).

5.1 pSlim vs. DSI

Mit der contxt-Bibliothek stehen DROPS-Anwendungen selektiv pSLIM und DSI als Übertragungsprotokolle zur Verfügung. Damit bot sich ein Leistungsvergleich der beiden verwendbaren Protokolle an. Allerdings war ein direkter Vergleich nicht möglich, da pSLIM eine synchrone und DSI eine asynchrone Übertragung darstellen. Gewisse Teilbereiche ließen sich jedoch gegenüberstellen.

Gemessen wurde in erster Linie die Verzögerungszeit vom Aufruf der jeweiligen contxt-Funktion bis zum Beenden des Darstellens. Großen Einfluss auf das Zeitverhalten hatte hierbei die Farbtiefe, da sie festlegt, wie viel Bitdaten in den Framebuffer zu kopieren sind. Zusätzlich ließ sich bei Verwendung des pSLIM-Protokolls auch der Zeitanteil für die IPC-Kommunikation sowie des eigentliche Darstellen messen. Der IPC-Anteil lag bei allen Messungen um die 110 μs.

Folgende Befehle der contxt-Bibliothek dienten als Messgrundlage:

printf gibt eine Zeichenkette aus (die Zeichenkette bestand bei der Messung aus 80 Zeichen)

putchar gibt ein Zeichen aus

contxt_clrscr löscht den kompletten Bildschirm

Die Aufgabe dieses Befehls ist den kompletten Bildschirmbereich mit der aktuellen Hintergrundfarbe zu füllen. An dieser Stelle sei erwähnt, dass die 16-Bit `fill`-Operation mittels Assemblercode optimiert wurde und somit nicht ganz mit dem Ergebniss für 24-Bit Operationen verglichen werden kann.

Im Vergleich der Gesamtdauer der Abarbeitung des Befehls war zu erwarten, das DSI aufgrund des Synchronisations-Overheads bei geringem Datenaustausch keine wesentlichen Performanzvorteile bringt. Die Messungen bestätigten diese Annahme. In Tabelle 5.1 werden die einzelnen Messergebnisse dargstellt.

Befehl	Protokoll	Anteil	Farbtiefe	
			16 bit	24 bit
putchar	pSlim	Gesamt	143 μs	171 μs
		Zeichnen	30 μs	57 μs
	DSI	Gesamt	251 μs	288 μs
printf (80 Zeichen)	pSlim	Gesamt	1,8 ms	4,0 ms
		Zeichnen	1,7 ms	3,6 ms
	DSI	Gesamt	2 ms	4,0 ms
clrscr	pSlim	Gesamt	32 ms	76,8 ms

Tabelle 5.1: Ausführungszeit einzelner contxt-Befehle

5.2 Copy vs. Redraw

Eine Funktion, die sehr häufig auftritt ist das Scrolling. Hierbei muss der komplette Bildschirminhalt, um eine gegebene Anzahl von Zeilen versetzt, neu gezeichnet werden. Den Anwendungen stehen dabei zwei Verfahren zur Auswahl. Zum einen kann auf dem Framebuffer der Grafikkarte direkt umkopiert werden. Zum anderen ist es möglich, jede Textzeile noch einmal neu darzustellen. Da die verwendete Grafikkarte keine Hardwareunterstützung anbietet, ist zu erwarten, dass ein Neuzeichnen jeder Bildschirmzeile performanter ist, als ein Kopieren im Framebuffer. Beide Verfahren wurden mit unterschiedlichen Bildschirmauflösungen gemessen. Die in Tabelle 5.2 aufgeführten Messergebnisse zeigen, dass ein Kopieren innerhalb des Framebuffers ohne Harwareunterstützung in der Tat teurer ist.

Methode	640x480x16	800x600x24	1024x768x24
redraw	71 ms	203 ms	279 ms
copy	95 ms	664 ms	1,1 s

Tabelle 5.2: Scrolling

5.3 Framebuffer

Das im vorherigen Abschnitt gezeigte Verhalten des Kopierens soll mit einigen Messungen noch einmal verdeutlicht werden. Die folgenden Tabellen vergleichen die Parameter (z. B. Zy/Z[1]) für einen schreibenden, lesenden sowie kopierenden Zugriff auf den gesamten Framebuffer ohne Hardwareunterstützung auf verschiedenen Rechnern.

Auflösung	Schreiben			Lesen			Kopieren		
	Durchsatz in Mb/s	Zy/Z	Dauer in ms	Durchsatz in Mb/s	Zy/Z	Dauer in ms	Durchsatz in Mb/s	Zy/Z	Dauer in ms
640x480x16	48,96	3,5	11	8.17	21.1	71	6.26	27,5	93
640x480x24	48,96	5,3	17	8,17	31,6	107	6.25	41,3	140
800x600x16	48,96	3,5	18	8.09	21,3	113	6,25	27,5	146
800x600x24	48,96	5,3	28	7,76	33.3	177	6,21	41,6	221
1024x768x16	48,96	3,5	30	8,12	21,2	184	6,21	21,8	241
1024x768x24	48,96	5,3	45	7,91	32,7	285	5,94	43,6	379

Tabelle 5.3: Pentium 90, ATI Rage XL, VBE version 2.0

Auflösung	Schreiben			Lesen			Kopieren		
	Durchsatz in Mb/s	Zy/Z	Dauer in ms	Durchsatz in Mb/s	Zy/Z	Dauer in ms	Durchsatz in Mb/s	Zy/Z	Dauer in ms
640x480x16	51,29	60	11	8.21	375	71	6.64	465	88
640x480x32	51,21	120	22	8,21	751	142	6.59	937	178
800x600x16	51,25	60	17	8.21	375	111	6,60	467	138
800x600x32	50,10	121	35	8,19	752	223	6,61	934	277
1024x768x16	51,13	60	29	8,21	376	182	6,59	468	228
1024x768x32	50,96	121	59	8,18	753	366	6,59	937	456

Tabelle 5.4: Pentium IV 1600 Mhz, Matrox Millennium G550, VBE version 3.0

Der große Unterschied in den Zyklenanzahl pro Zeichen ist mit der unterschiedlichen Taktrate der Testrechner zu begründen. Die von der ATI RAGE XL Grafikkarte verwendete VBE Version 2.0, bietet eine maximale Farbtiefe von 24 Bit. Version 3.0 hingegen kann bis zu 32 Bit Farbtiefe darstellen. Ein direkter Vergleich ist somit nur im 16-Bit Farbmodus möglich.

[1] Zyklen pro Zeichen

Kapitel 6

Zusammenfassung und Ausblick

Im Rahmen dieses Beleges wurde eine Schnittstelle zur Ein- und Ausgabe von Zeichendaten sowohl für L⁴Linux als auch für DROPS-Anwendungen entwickelt und implementiert. Das DROPS-Konsolensystem ist zu diesem Zweck geringfügig erweitert und angepasst worden.

Das Ziel, eine geeignete Schnittstelle zur Übertragung von Grafikdaten zu entwickeln, konnte aufgrund der Komplexität nicht erfüllt werden. Für weiterführende Arbeiten sollte dies jedoch ein interessante Aufgabe darstellen.

Mit der Entwicklung einer DROPS-C-Biblothek könnte die Bereitstellung einer nutzerfreundlichen Schnittstelle zur Ein- bzw. Ausgabe und das eigentliche Einbringen in den Framebuffer gekapselt werden. Die dafür benötigte read/write-Schnittstelle ist in der contxt-Biblothek bereits implementiert worden.

In Hinblick auf die bescheidene Performance der Ausgabeoperationen ist Hardwareunterstützung unumgänglich. Für die weitere Arbeit am DROPS-Konsolensystem sollte daher eine vollständige Portierung des Grafikartentreibers im Vordergrund stehen.

Anhang A

Glossar

Clipping Abschneiden von nicht sichtbaren Bereichen.

Framebuffer ein Bereich des Speichers der Daten beinhaltet. Meist der Speicher einer Grafikkarte auf dem Adapter von dem das Monitorbild aufgefrischt wird.

history buffer puffert Informationen die nicht mehr auf dem Bildschirm sichtbar sind.

IPC (InterProcess Communication; engl. Interprozeßkommunikation) Mechanismus des Datenaustausches zwischen verschiedenen Programmen (in unterschiedlichen Adreßräumen)

history buffer [eng.] Zwischenspeicher, puffert Informationen die nicht mehr auf dem Bildschirm sichtbar sind.

scrolling [eng.] Abrollen, mit Scrollen bezeichnet man den Vorgang, in dem man den Bildschirminhalt scheinbar nach links/rechts oder oben/unten verschiebt, um andere Inhalte des Bereichs zu sehen.

Terminal Ein/Ausgabegerät für Computer mit Tastatur und Monitor

Textrendering umwandeln von Zeichendaten in Bitmapdaten

Literaturverzeichnis

[1] J. Liedtke. L4 reference manual (486, Pentium, PPro). Arbeitspapiere der GMD No. 1021, GMD — German National Research Center for Information Technology, Sankt Augustin, September 1996. Also Research Report RC 20549, IBM T. J. Watson Research Center, Yorktown Heights, NY, September 1996.

[2] Brian K. Schmidt, Monica S. Lam, and J. Duane Northcutt. The interactive performance of SLIM: a stateless, thin-client architecture. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP)*, volume 34, pages 32–47, December 1999.

[3] Christian Helmuth. Ein Konsolensystem für DROPS. Großer beleg, TU Dresden, August 2000.

[4] David S. Lawyer. *Text-Terminal-HOWTO*. http://www.linuxdoc.org, Januar 2002.

[5] Geert Uytterhoeven. *The Linux Frame Buffer Device Subsystem*. http://www.linux-fbdev.org, 1999. Linux Expo.

[6] Andreas Beck and Steffen Seeger. *The General Graphics Interface*. http://www.ggi-project.org.

[7] Steffen Seeger. *The Kernel Graphics Interface*. http://kgi.sourceforge.net.

[8] Jork Löser, Lars Reuther, and Hermann Härtig. A streaming interface for real-time interprocess communication. Technical Report TUD-FI01-09-August-2001, TU Dresden, August 2001. Available from URL: `http://os.inf.tu-dresden.de/ jork/`.

[9] Mathias Noack. Ein Textkonsolensystem für DROPS. Komplexpraktikum, TU Dresden, April 2001.