User Level Device Driver am Beispiel von TCP

Christian Hennrich

September 17, 2004

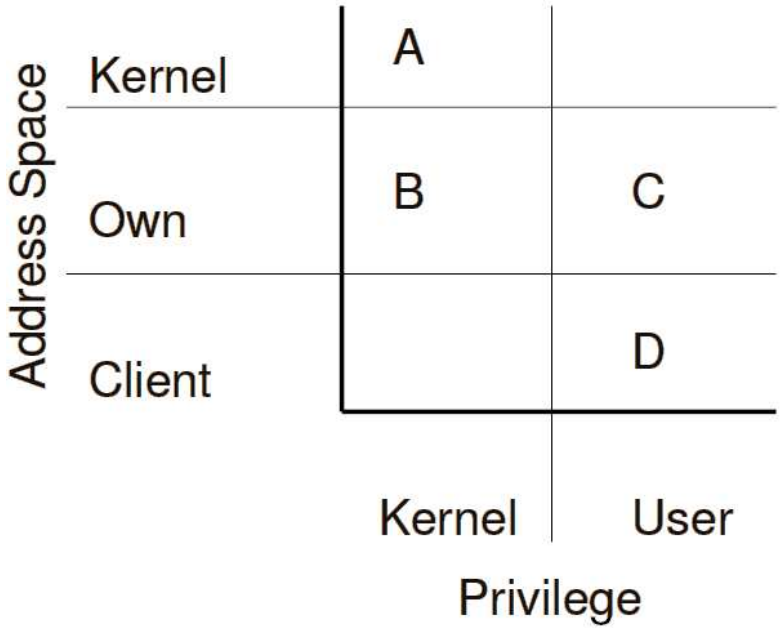
Motivation für Userlevel Device Driver

- Modularität, leichterer Austausch/Erneuerung von Komponenten.
- Sicherheit, Neustart einzelner Treiber nach Fehlfunktion. Fehlertoleranz.
- Stabilität gegenüber anderen Treibern
- Steuerung, Überwachung von Treibern. Informationen im Userbereich lesbar.
- Kleinere Mikrokernel.

Probleme von Userlevel Device Driver

- Schlechte Performance
  - Latenzzeiten größer
  - CPU Idle Time gering
- IPC, mehr Kommunikationsaufwand, langsam
- Sicherheitsaspekt von Shared-Memory, Allokierung
- Nur bedingter DMA Zugriff über andere Server.
- Crossing of Protection Boundaries.

Übersicht



- A normaler Linux Device Driver
- B separater Kerneladressspace, Kernel Priviligien
- C linked Library (in-process)
- D separater Prozess (IPC)

Existierende Systeme

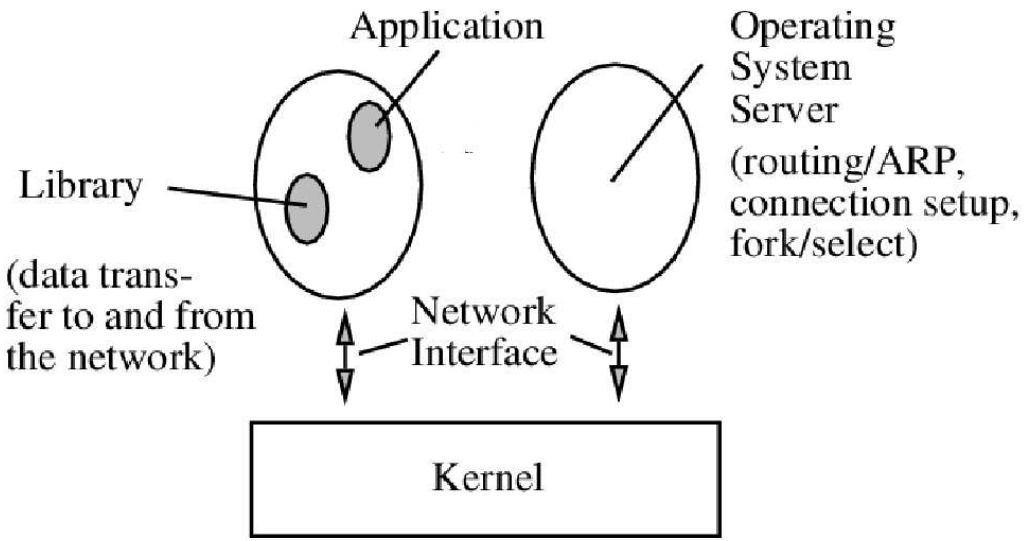
Monolithische Kernel

- U-Net, Shared-Memory
- HP-UX, Linked-Library

Multiserver Betriebssysteme

- Mach Linked-Library
- Mach Userlevel TCP-Stack
- I/O oriented IPC
- Sawmill

Mach Userlevel TCP-Stack

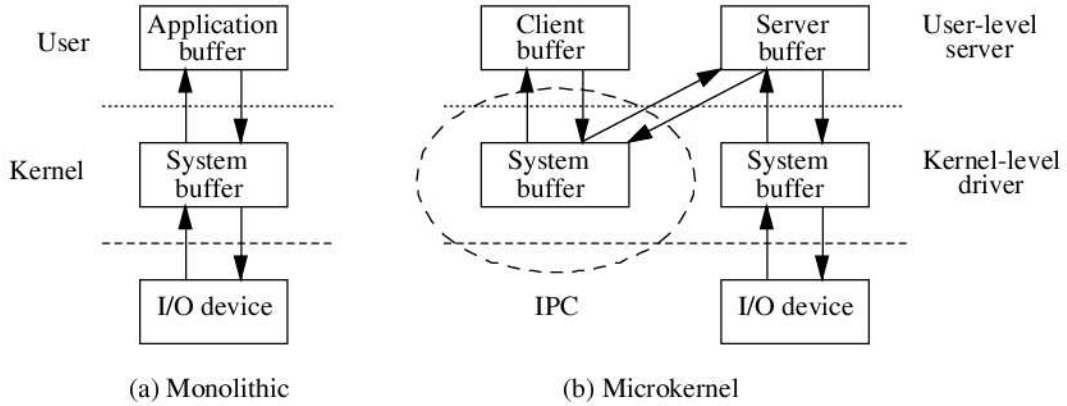


- Multiserver Idee
- Operating System Server
- Linked Library
- Pro Paket ein IPC

Optimierungen

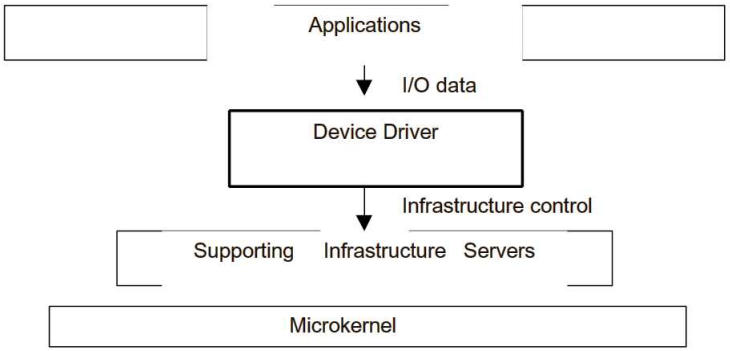
- Mehrere Paket pro IPC
- Shared-Memory
- Shared-Buffer (Sockets)

I/O oriented IPC



- Aufbau wie Mach Userlevel TCP-Stack
- Crossing of Protection Boundaries
- Kommunikation über kernelkontrollierte Buffer
- Emulated Copy (Einmappen der gewünschten Seite)

Sawmill



- Verwendung von Linuxcode
- Linuxsubsysteme als eigenständige Server
- Basisdienste (Naming, Emulation Librarys, Memorymanagement)
- Device Driver Manager (DDM), Verwaltung der Systemressourcen (Speicher, IRQs)
- Datenaustausch über Buffer im Shared-Memory
- Übergabe der Bufferreferenzen mittels IPC

Durch IPC schlechte Performance ➡ Optimierung

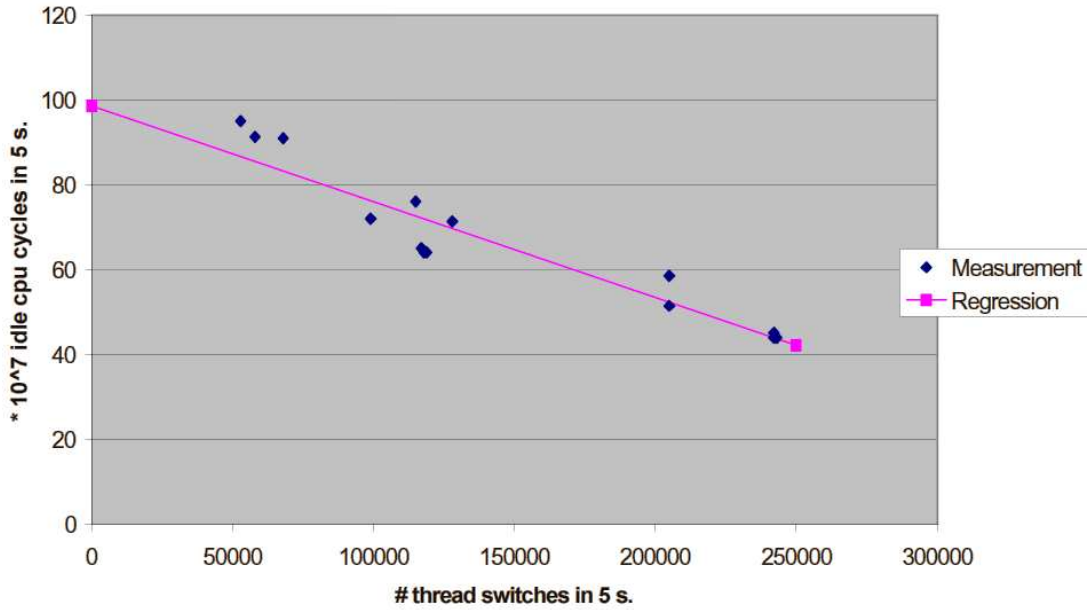
Vergleich Linux vs Sawmill

Results

Setup	Linux idle time confidence interval		Throughput	Sawmill idle time Confidence interval		Throughput
Unit	*[idle cycles]*		*[Mbit/s]*	*[idle cycles]*		*[Mbit/s]*
TCP Transmit	68.5 %	68.7 %	82.9	29.8 %	30.0 %	83.1
TCP Receive	70.6 %	71.0 %	62.7	26.9 %	27.3 %	63.2

- Gleicher TCP-Stack, Messung auf Pentium II 300MHz
- Unterschiedliche CPU-Auslastung
- Ähnliche Bandbreite

Kontextwechsel als Grund für die hohen Kosten



- Variation der Kontextwechsel
- Datenfluss bleibt gleich
- Lineares Verhalten zw. Kontextwechsel und CPU-Idle Time

Kosten

- TCP Transmitting mit IPC Saving Methode
- TCP Receiving mit IPC Saving Methode
- TCP Transmitting mit 1 Paket pro IPC
- TCP Receiving mit 1 Paket pro IPC

Setup	# packets	# IPCs	# IRQs	Busy time
Unit	*/ 5 [s]*	*/ 5 [s]*	*/ 5 [s]*	*[s] / 5 [s]*
Setup 1 (Tx)	57100	3550	24250	1.98
Setup 2 (Rx)	43420	12950	37490	2.75
Setup 3 (Tx)	57880	24510	25540	2.83
Setup 4 (Rx)	42600	42300	32960	3.5

- 14 μs Verarbeitungszeit pro Paket
- 34 μs pro IPC
- 44 μs pro IRQ

Kosten der IPCs

Individuelle Kosten pro Kontextwechsel, absteigend

- IRQ IPCs
- Send IPC
- Delay IPC, dann Send IPC
- Delay IPC

Vermeidung von Kontextwechsel
Benutzung weniger CPU-intensiver Kontextwechsel

Überblick

- Allgemein programmierbare Schnittstelle (API)
- Treiber: Server
- Direkte Kommunikation mit der Hardware (I/O,IRQ)
- Daten-Multiplexing
- Client: userlevel Kommunikationsprogramme
- Device Driver Manager DDM (Naming,Shared-Memory)
- Aufteilung in Mangement- und Kommunikationsteil
- Betrachtung des Kommunikationsteil
- Shared-Memory (Data Space Manager)
- IPC zur Kommunikation

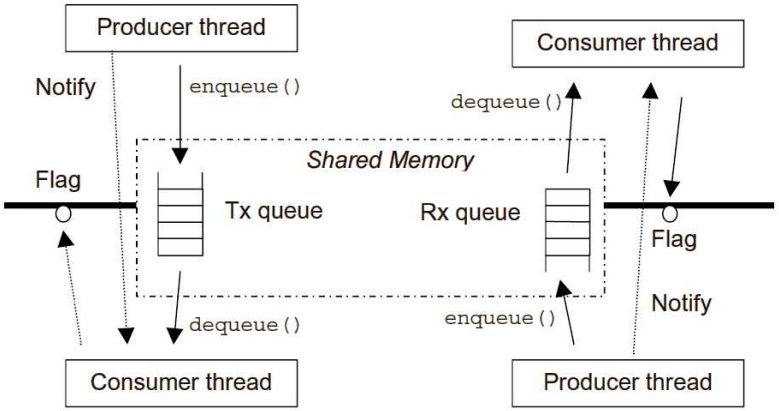
Voraussetzungen

Performancesteigerungen nicht zu Lasten der Sicherheit
Richtlinien:

1. Objekte des Designs dürfen nicht in der Lage sein sich gegenseitig zu beschädigen, weder zufällig noch gewollt.
2. Clients dürfen nicht auf die Daten anderer Clients zugreifen, weder direkt noch indirekt (Benutzung des Treibers).

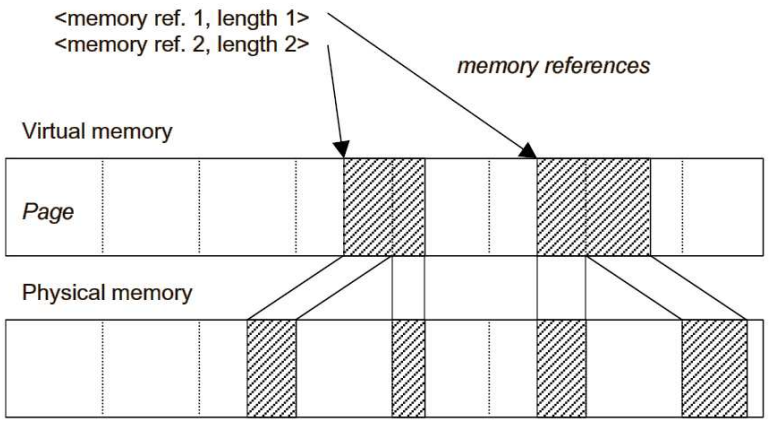
- Verletzung bedeuted Beendigung der Kommunikation
- Nur **ein** Client und Treiber pro Shared-Memory Bereich
- Bufferinhalt muss selbstständig geprüft werden.

Kommunikation und Synchronisation



- 2 FIFO Queues
- Kommunikation über IPC und Flags (Shared-Memory)
- Datenaustausch über Shared-Memory
- Blockade der L4-Threads über IPC

Buffer 1.Teil



- Logischer Buffer besteht aus Buffer-Metadaten und Buffern
- Speicherung in den FIFO TX und RX Queues
- Zusammenhängend im Virtual Memory
- Aufgesplittet im Physical Memory

Buffer 2.Teil

Shared Memory Umgebung

- Allokierung beim Senden mit `alloc()`
- Freeing beim Empfangen mit `free()`
- 3 Arten von Speicherblöcken: Free, Client, Server
- Vereinbarung, angelegt Speicher einer FIFO gehört korrespondierender Seite

Direkt Memory Access DMA

- Hardware greift nur direkt auf Speicher zu
- CPU verwaltet virtuellen Speicher
- Einführung des Data Space Managers (DSM)
- DSM garantiert Pinning (Mapping, Dauer)
- DSM übernimmt Adresstranslation

Protection

- Korrekt arbeitender DSM garantiert Zugriff
- Shared-Memory fehlerhaft
- Keine Übertragung auf andere Shared-Memorys
- Beendigung der Kommunikation
- 2 Arten von Fehler
  - Datenfehler
  - Progressfehler

Protection

Datenfehler

- Kopieren der Daten und Überprüfung
- Arbeiten auf Kopie
- Pointerüberprüfung mittels DSM
- Datenüberprüfung (Pointer, Länge)

Progressfehler

- Überprüfung auf Endloschleifen
- Kopieren der Abbruchbedingung

Consumer IPC Saving

Queue Überprüfung durch IRQ

- NIC Benachrichtigung des Treibers
- Treiber setzt Flag, ob Client Benachrichtigung schicken soll
- Driver kennt Anzahl zukünftiger IRQs
- Preemptionproblem bei einem IRQ
- Nur ein IRQ ⇨ Flag setzen

Periodisches Queue Polling

- Kein IRQ zum Überprüfen der Queues bei Clients
- Systemtimer veranlassen Überprüfung
- Intervalllängewahl ⇨ Latenz
- Aufwecken eines Threads <IPC pro Paket

Producer IPC Saving

Verzögerte Benachrichtigung

- IPC nach letztem Paket
- Nur für Client, Treiber kann kein Voraussage machen
- Einführung eines Timers für Treiber
- Änderung der L4 Timer

Periodische Benachrichtigung

- Benachrichtigungs IPC nach max. Latenzzeit
- Ähnlich wie Periodisches Queue Polling
- Zusätzlich ein IPC
- Aufwecken eines Threads + IPC <IPC pro Paket

CPU-Idle Time nach der Optimierung

	Interval Queue Polling & Imprecise Timers		Linux 2.2.5
Benchmark	CPU Idle	Context sw.	CPU Idle
Transmit	63.30 %	53000	72 %
Receive	59.50 %	175000	70 %