3. Neue OS-Architektur?

2

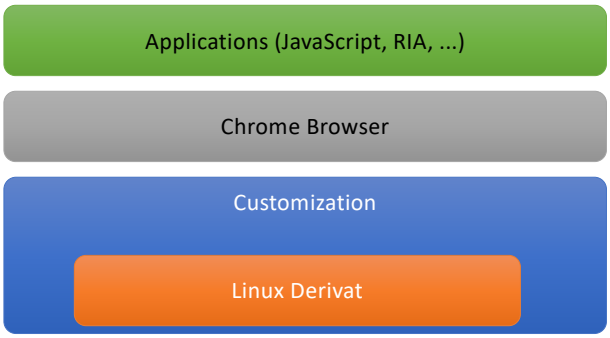
Herausforderungen

- ManyCores

3

Chrome OS + Chromium OS

- Kategorie „Just enough operating system“
- Chromo OS = Google „Produkt“
- Chromium OS = Open Source Projekt



Microsoft Singularity

- Forschungsprojekt von Microsoft Research
  - 2004 – ~2009
- Design Prinzipien
  - Kompletter Neubeginn (Name ☺)
  - Keine Abwärtskompatibilität
  - Fokus auf Zuverlässigkeit
  - Typsichere Sprachen und Softwareverifikation
  - Microkernel-Architektur

wikipedia

Adressräume

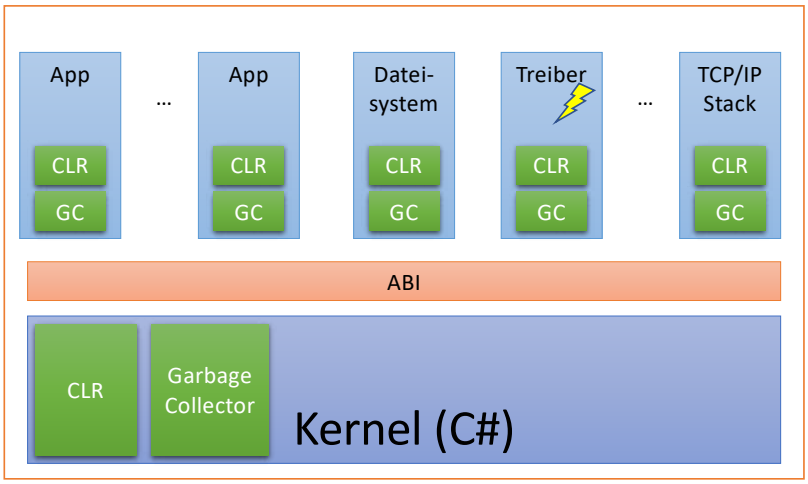
- Isolation von …
  - Verschiedenen Prozessen
  - Kernel und Prozessen
- Grund?
  - Unmanaged Code
    - C / C++, Assembler, …
  - Nicht typsichere Sprachen
  - Potenziell fehlerhafte Verwendung von Pointern

Folgen

- Komplexe MMUs
  - TLB
  - Mehrstufige Seitentabellen
  - Deskriptoren
- Memory Management im OS

8

Singularity Ansatz



Software-isolierte Prozesse (SIPs) I

- Alles im gleichen Adressraum!
  - Kernel, Treiber, Applikationen
- Isolation
  - Typsichere Sprache (keine Pointer)
  - Alle Referenzen explizit und typisiert
  - Verifikation während der Übersetzung
  - Isolation ist formal beweisbar!

Software-isolierte Prozesse (SIPs) II

- Keine Isolation auf HW-Ebene
  - Nicht notwendig
  - Kein Adressraumwechsel
  - Keine kalten Caches / TLBs, …

➔ Microkernel-Ansatz wird effizient!

Software-isolierte Prozesse (SIPs) III

- Verifikation zur Übersetzungszeit
  - Deshalb (momentan) …
    - Keine Code-Generierung zur Laufzeit
    - Kein dynamisch nachgeladener Code

Interprozess Kommunikation

- Komplette Isolation?
- Interaktion über spezielle Kommunikationskanäle
  - Nachrichtentypen
  - Abfolge per Kontrakt spezifiziert
  - Compiler überprüft korrekte Verwendung
- Abhängigkeiten
  - Vollständig über Kontrakte spezifiziert
  - Zwischen allen SIPs
    - Treiber, Applikationen, Kernel, Dateisystem, …

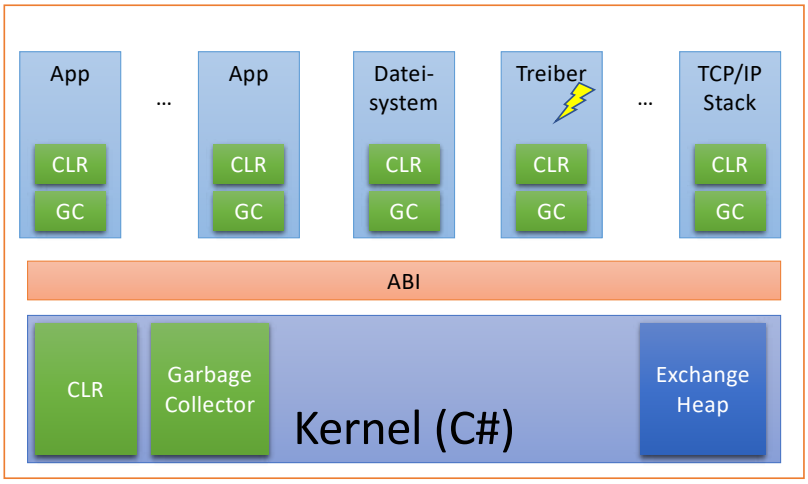
Sing#

- C#-Dialekt Sing#
- Kommunikation und Kontrakte auf Sprachebene
  - Definition von Endpunkten
  - Empfang von (mehreren) Nachrichten
  - Senden von Nachrichten
  - Zustandsmodell

```
public contract TcpConnectionContract {
    in message Connect(uint dstIP, ushort dstPort);
    out message Ready();

    state Start : Ready! -> ReadyState;
    state ReadyState : one {
        Connect? -> ConnectResult;
        BindLocalEndPoint? -> BindResult;
        Close? -> Closed;
    }
    state BindResult : one {
        OK! -> Bound;
        InvalidEndPoint! -> ReadyState;
    }
    in message Listen();
    state Bound : one {
        Listen? -> ListenResult;
        Connect? -> ConnectResult;
        Close? -> Closed;
    }
    …
}
```

Technische Umsetzung



Exchange Heap

- Durch Kernel verwaltete Datenstruktur
  - Enthält typisierte Nachrichten
  - Kernel erlaubt und entzieht Zugriff auf Teile des Heaps
- Austausch einer Nachricht
  - Referenz auf Nachricht an Kernel übergeben
  - Kernel benachrichtigt Empfänger und übergibt Referenz

  ➔ Simple „Pointer“-Operation
- Invariante
  - Immer nur ein Prozeß hat Referenz auf Teil des Exchange Heaps
  - Verifikation durch Compiler auf Basis des Kontrakts

Landläufige Meinung

In C/C++ geschriebene Software läuft schneller als Software, die in C# oder JAVA geschrieben wurden!

18

Auf den zweiten Blick …

- Performancevorteile
  - IPC ohne Kontextwechsel
  - Kernelaufrufe ohne Kontextwechsel
  - Keine Runtime Checks notwendig
    - Checks einmalig bei der Übersetzung
- Erste Benchmark-Ergebnisse
  - Faktor 5 – 10 schneller als aktuelle Betriebssysteme (FreeBSD, Linux, Windows)

Microsoft Midori

Microsoft Midori

- Forschungsprototyp mit Praxisbezug
  - Sogenanntes „Incubation“-Projekt
  - Basiert auf Singularity
  - Geheimnisumwittert
- Spekulationen
  - Zukünftige Windows-Version?
  - Bislang wenig technische Details bekannt
- Trend zu formalen Methoden


Microsoft's Midori operating-system skunkworks project soldiers on

It Lives! A Prototype in Use

Formal language models a prototype C# extension in active use at Microsoft.

- Some differences from formal system
  - e.g. first-class tasks, unstrict blocks

Midori Programmiermodell

- „Concurrency“
  - Asynchrone OS API
  - Kommunikation über asynchrone Nachrichten
  - Erlaubt bessere MultiCore Skalierbarkeit
  - siehe z.B. auch MS Robotics Studio
- „Distributed Concurrency“
  - Einbeziehung entfernter Komponenten
  - → Cloud Computing
  - Programmiermodell unterstützt Nachrichtenverlust, Latenzen, sporadische Konnektivität

Virtualisierungsaspekte

- Virtualisierbarkeit des Kernels
  - Nativ ausführbar auf x86, x86 und ARM
  - Ausführung im Hypervisor
    - typsichere Interaktion zwischen Hypervisor und Kernel
  - Ausführung in einem Windows-Prozess

Quellen

- Artikel
  - Galen C. Hunt, James R. Larus, Singularity: Rethinking the Software Stack, ACM SIGOPS Operating Systems Review, vol. 41, no. 2, pp. 37-49, Apr. 2007
  - Mark Aiken et al., Deconstructing Process Isolation, in ACM SIGPLAN Workshop on Memory Systems Performance and Correctness, pp. 1-10, ACM, San Jose, CA, Oct. 2006
  - Galen Hunt et al., An Overview of the Singularity Project, no. MSR-TR-2005-135, pp. 44, Microsoft Research, Oct. 200
- Singularity als Source Code und bootbares Image
  - http://www.codeplex.com/singularity
- Channel 9 Videos: http://channel9.msdn.com