KVM

Dr.-Ing. Volkmar Sieh

Department Informatik 4
Verteilte Systeme und Betriebssysteme
Friedrich-Alexander-Universität Erlangen-Nürnberg

WS 2019/2020

Fragestunde

Fragen zu den Aufgaben?

Übersicht

Entwicklung

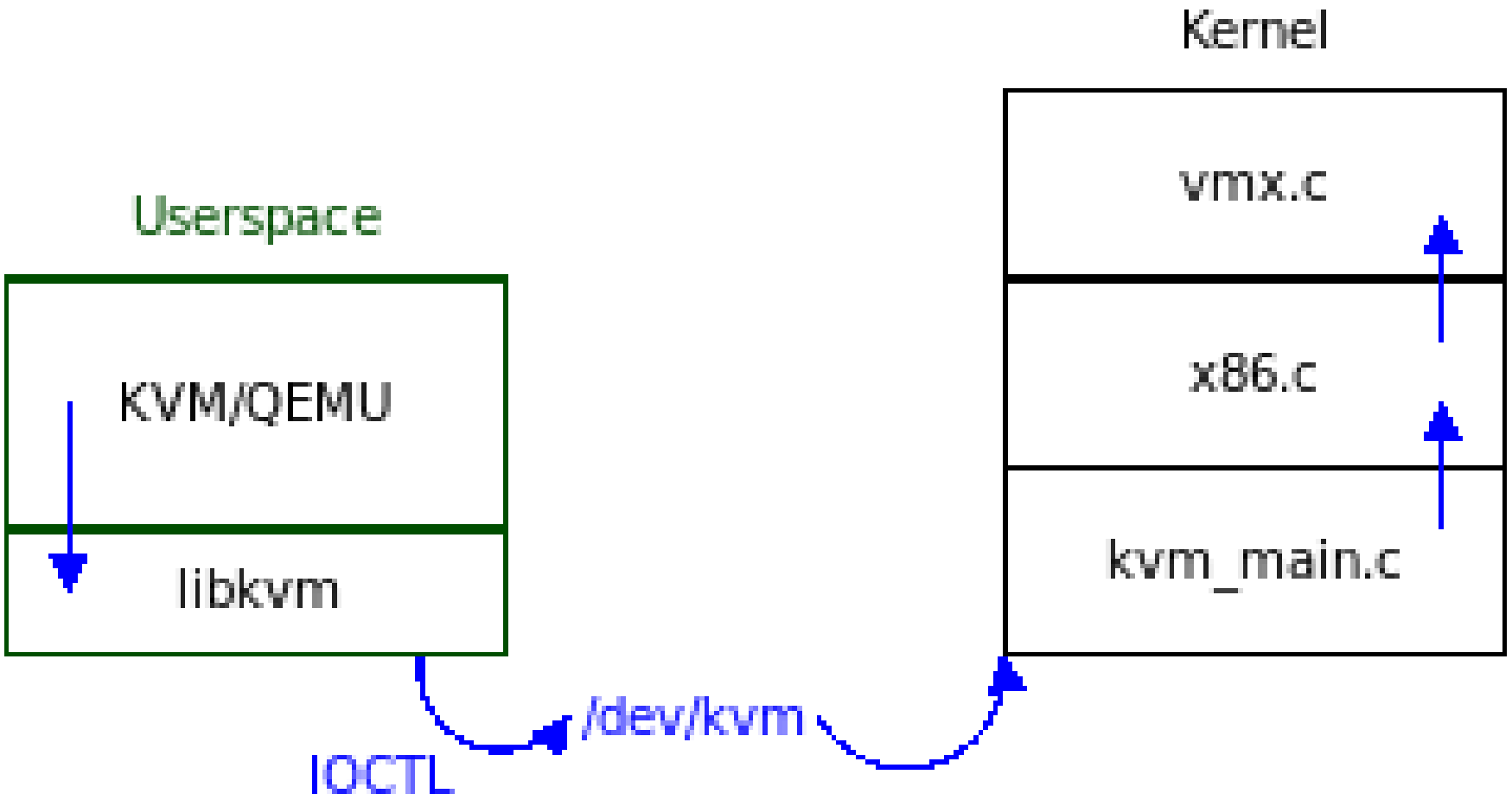
- 2007 „aus dem Nichts“ aufgetaucht
- Entwickelt von Qumranet (welches später von RedHat aufgekauft wurde)
- Teil des offiziellen Kernels seit 2.6.20
- Userspace: Fork von QEMU
- derzeit sehr populär bei den Distributoren
- Support für x86
- In Bearbeitung (evtl. schon lauffähig?):
  - PowerPC (IBM)
  - S390 (IBM)
  - IA64 (Intel)

Übersicht

Kernel-based Virtual Machine, Bestandteile

- Kernel-Modul
  - Architekturunabhängiger Teil (`virt/kvm/*`)
    - Kommunikation mit Userspace
    - MMIO Verwaltung
  - x86-Spezifischer Teil (`arch/x86/kvm/*`)
    - Hardware-Schnittstelle (`svm.c`, `vmx.c`)
    - Emulation (`x86_emulate.c`)
    - Interface für Shadow-Page-Table (`mmu.c`)
    - Emulierter PIT und PIC `i8254.c`, `i8259.c`
- Userspace
  - QEMU-Fork, welcher Kernel-Modul benutzt
  - libkvm als *glue* und Frontend zur Kommunikation mit dem Kernel
- Kommunikation über `/dev/kvm` Gerät mittels `ioctl`

Beispiel


Kernel
vmx.c
x86.c
kvm_main.c
Userspace
KVM/QEMU
libkvm
/dev/kvm
IOCTL

Userspace → Kernelmodul

Interface

- VM Initialisieren
- Logischen Prozessor initialisieren
- Speicherbereich registrieren
- Register auslesen/setzen
- Code ausführen (→ `VMRUN`)
- Grund für Rückkehr ermitteln
- IRQ injizieren
- rudimentärer Support für Paravirtualisierung:
  - PCI-Gerät in Gast benutzen
  - IRQ durch Gast behandeln

Userspace

libkvm – Glue für `ioctl`

- Vereinfachung der Speicherverwaltung
- Hauptschleife
- Callbacks für MMIO
- Callbacks für besondere Punkte in Hauptschleife
- Kapselung der `ioctl`-Mechanismen in Funktionen

Kernel-Modul

Instanziierung

- Prozessorabhängiges Modul als Einstiegspunkt (`vmx.c` oder `svm.c`)
- übergibt Callbacks an architekturunabhängiges Modul (`kvm_main.c`)
- dieses ruft die `init`-Funktionen der Komponenten auf (z.B. `kvm_arch_init` in `x86.c`)
- welche weitere Callbacks registrieren

Kernel-Modul

Architekturunabhängig: `kvm_main.c`

- Registrieren des `kvm`-Geräts
- Einstiegspunkt für `IOCTL`
- Speicherverwaltung
- Locking
- Kapselung von architekturabhängigen Funktionen

Kernel-Modul

Architekturabhängig: `x86.c`

- Implementierung der `IOCTL's`
- Frontend für:
  - Speicherverwaltung/MMU (Shadow Page Tables)
  - Prozessorspezifische Funktionen
  - Emulator

`mmu.c`

- Shadow Page Tables
- Reverse Page Table für Gast
- Diverse Funktionen zur Adressumrechnung
- absolut nicht trivial!

Kernel-Modul

Emulation `x86_emulate.c`

1. Instruktion wird dekodiert
2. ggf. wird MOD/RM und SIB geholt
3. Operation wird
   - in manchen Fällen emuliert
   - meistens ausgeführt (!)
4. ggf. wird das Ergebnis zurückgeschrieben

Sofern nötig, wird außerdem

- Userspace MMIO ausgeführt
- Userspace IO ausgeführt

Zusammenfassung

KVM

- optimale Performanz bei rechenintensiven Tasks
- Langsam bei häufigen Wechsel in den Userspace
- einfaches Interface mittels `libkvm`
- leider sehr anfällig für Fehler aus Userspace
- schwer zu debuggen

Organisatorisches

Nächste Woche ...

noch einmal? Wenn ja, welches Thema interessiert Euch?

Organisatorisches

Danke

- fürs Mitarbeiten
- fürs Zuhören
- für die **tollen** Lösungen!

Ausblick

Bachelor-/Masterarbeit/Masterprojekt im Bereich FAUmachine?