Xen Workshop

Jürgen Groß - jgross@suse.com

Agenda

- Xen Überblick
- Architektur von Xen im Detail
- Einsatzkonfigurationen von Xen
- Basismechanismen von Xen
- Konfigurieren und Bauen von Xen
- Abläufe im Detail
- Debugging von Xen
- Live Vorführung
- Zusätzliche Informationen

Xen Überblick

Xen Überblick

- Virtualisierung Grundlagen
- Xen Grundlagen
- Architektur Überblick von Xen
- Sicherheitskonzept

Virtualisierung Grundlagen

- Wikipedia: “*Virtualisierung bezeichnet in der Informatik die Nachbildung eines Hard- oder Software-Objekts durch ein ähnliches Objekt vom selben Typ mit Hilfe einer Abstraktionsschicht.*”
- Hier: Virtualisierung eines Computer-Systems (Prozessor, Speicher, I/O-System)
- Warum?
  - Bessere Hardware Ausnutzung (mehrere Instanzen laufen virtualisiert auf einem System)
  - Flexibilität bei Zuteilung von Hardware-Ressourcen an einzelne virtualisierte Instanzen (dynamische Vergrößerung oder Verkleinerung von Speicher, Prozessoranzahl)
  - Verschiebung der virtualisierten Instanzen auf andere Hardware im laufenden Betrieb (“*live migration*”)
- Heutzutage mit Hilfe von Virtualisierungsfunktionen in der Hardware, früher teilweise reine Software-Lösungen
- Wichtig: virtualisierte Instanzen (“Gäste”) müssen voneinander abgeschottet werden

Virtualisierung Grundlagen

- Softwareschicht zur Realisierung der Abstraktionsschicht: **Hypervisor**
- Aufgaben des Hypervisors:
  - Zuteilung der Ressourcen an die Gäste
  - Realisierung der Abschottung zwischen den einzelnen Gästen
  - Bereitstellung der Ablaufumgebung (die eigentliche Virtualisierung)
- Unterschieden werden Hypervisor vom Typ 1 und Typ 2:
  - Typ 1: eigenständiger Mikro-Kernel mit nur minimaler Treiberausstattung; Verwaltung und I/O-Virtualisierung passieren in einer oder mehreren dezidierten Gästen
  - Typ 2: Hypervisor ist in Betriebssystem eingebettet (z.B. Linux Kernel); Verwaltung etc. passieren innerhalb dieses Betriebssystems

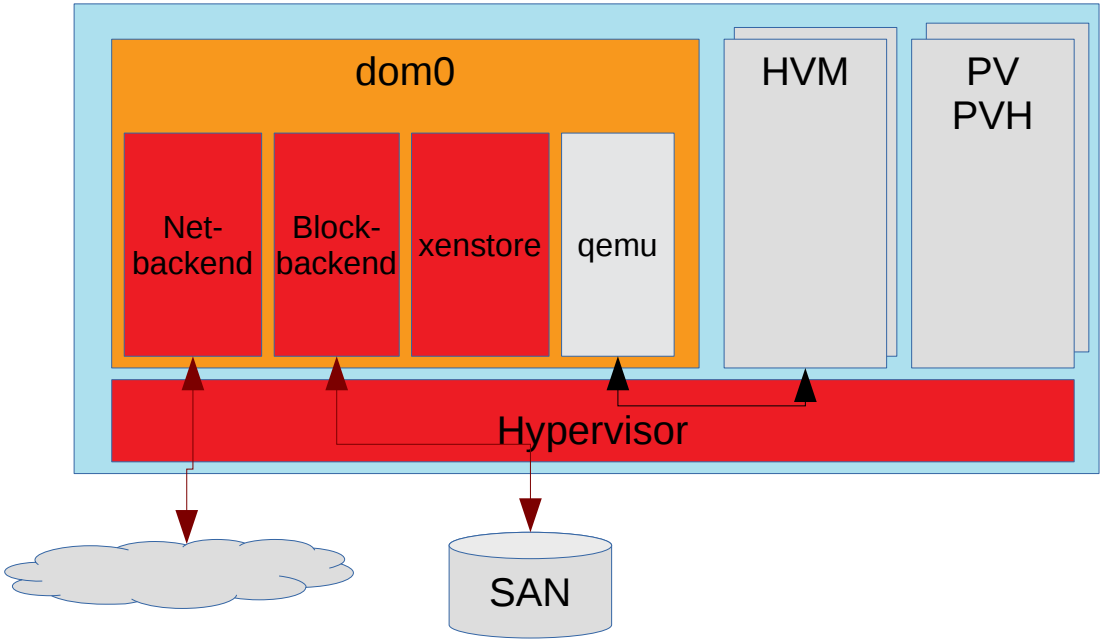
Xen Grundlagen

- Typ 1 Hypervisor
- Open Source (Lizenzen: GPL-V2, LGPL-V2.1, BSD)
- Programmiersprachen: C, Assembler, Python, OCaml
- Anfangs reine Software-Lösung, sehr gute Performance durch “*Paravirtualisierung*” (Gäste werden für Ablauf unter Xen speziell angepasst, um teure Emulation privilegierter Operationen zu vermeiden)
- Unterstützt x86 (64-bit), Arm (32- und 64-bit); RISCV und PowerPC in Arbeit; IA-64 und x86-32 nicht mehr
- Anwendungsgebiete: Embedded, User-Devices, Server, Cloud
- Gastsysteme mit expliziter Xen-Unterstützung: Linux, Windows, BSD, Solaris, BS2000, ...

Architektur Überblick von Xen

- Hypervisor kontrolliert Hardware-Zugriffe der Gäste
- Privilegierter Gast “*Domain-0*” wird direkt vom Hypervisor gestartet, enthält Xen tools zur Verwaltung des Systems
- Vollvirtualisierte Gäste (x86 HVM) haben jeweils einen qemu-Prozess in dom0 zur Emulation der Plattform (“*Device Model*”)
- I/O-Geräte werden normalerweise als “*PV-Devices*” in Form von PV-Treibern (Frontend im Gast, Backend in dom0) realisiert
- Konfigurationsdaten der Gäste sind im “*Xenstore*” abgelegt, auf den auch von den Gästen zugegriffen werden kann (Zugriffsrechte pro Eintrag konfigurierbar)

Architektur Überblick von Xen



Sicherheitskonzept

- Berechtigung der einzelnen Hypercalls (explizite Aufrufe des Hypervisors aus Gästen) kann konfiguriert werden (ähnlich wie Syscalls in SE-Linux)
- Zugriff auf Speicher eines anderen Gastes nur für dom0 (kann verboten werden), oder bei Erlaubnis durch den Gast selbst bei ausgewählten Speicherseiten (z.B. für I/O)
- Backends für PV-Geräte können auch in speziellen Gästen (“*Driver Domains*”) laufen (wird z.B. in Qubes-OS für Netzwerk verwendet)
- Manche Dienste in dom0 können in “*Stubdomains*” verlagert werden (qemu für HVM-Gäste, Xenstore) - “*Disaggregation*”

Architektur von Xen im Detail

Architektur von Xen im Detail

- Hypervisor
- Gast Typen
- Dom0
- DomUs
- Xen tools
- Xenstore
- PV Geräte-Treiber
- Virtio

Hypervisor

- System Initialisierung
- Speicherverwaltung
- Prozessor Verwaltung, Scheduling
- Interrupt Handling
- Gäste Verwaltung
- Zugriffskontrolle für Gäste
- Service Funktionen
- Live-Patching möglich (falls konfiguriert)

Gast Typen

- PV: nur x86
  - Funktioniert ohne spezielle Hardware-Unterstützung (reine SW-Lösung)
  - Gäste müssen modifiziert werden (Page-Table Handling, Interrupt Handling, …)
  - Gäste laufen im unprivilegierten Modus
- HVM: nur x86
  - Qemu in dom0 emuliert Plattform (Standard PC Peripherie)
  - Gäste können komplett unmodifiziert laufen, profitieren von PV-Gerätetreibern
- PVH
  - Normalerweise nur PV-Geräte (Ausnahme: direkt vom Gast verwaltete Geräte, z.B. per PCI-Passthrough oder Device-Tree)
  - Gast bestimmt ob 32- oder 64-bit (falls Hardware das erlaubt)

Dom0

- Direkt vom Hypervisor beim Boot des Systems gestartet
- Meistens Linux (FreeBSD hat auch Dom0-Unterstützung, historisch auch Solaris)
- X86: Standard als PV-Gast; PVH möglich, aber noch nicht komplette Funktionalität
- Hat normalerweise umfassende Privilegien
- Aufgaben (können teilweise in andere Gäste ausgelagert werden):
  - Hardware Systemverwaltung (über Hypervisor, z.B. Power-Management, Reboot, ...)
  - Konfiguration und Verwaltung von Gästen (über Xen tools)
  - I/O-Backends für Gäste

DomUs

- Normalerweise unprivilegiert (keine schädliche Beeinflussung anderer Gäste möglich)
- Sehen Konfiguration wie über Xen konfiguriert (Anzahl Prozessoren, Speicher, I/O-Geräte, Prozessor-Features)
- Profitieren von Anpassungen an Xen (PV-Erweiterungen, über Hypercalls nutzbar)

Xen tools

- Ablaufumgebung: dom0, User, meistens root
- Bestehen aus Bibliotheken und Programmen
- Version gekoppelt an Hypervisor (einige Bibliotheken haben ein stabiles Interface, nutzen intern aber low-level Bibliotheken von Xen ohne stabiles Interface)
- Funktionalität:
  - Starten/Stoppen von Xen Services (Xenstore, Xen Console Daemon)
  - Konfiguration und start/stop von Gästen
  - Info Funktionen
  - Debugging

Xenstore

- Zentraler hierarchischer Name-Value Store
- Entweder Daemon in dom0 oder eigene Domain ohne Peripherie
- Zugriffserlaubnis zu einzelnen Einträgen pro Gast einstellbar
- Unterstützt Transaktionen (mehrere Einträge können atomar gleichzeitig geändert werden)
- Über “*Watches*” kann man sich über Änderungen von einzelnen Einträgen benachrichtigen lassen (z.B. für Hotplug von PV-Geräten)
- Support sowohl im Linux Kernel als auch über Bibliothek im User-Mode

Xenstore

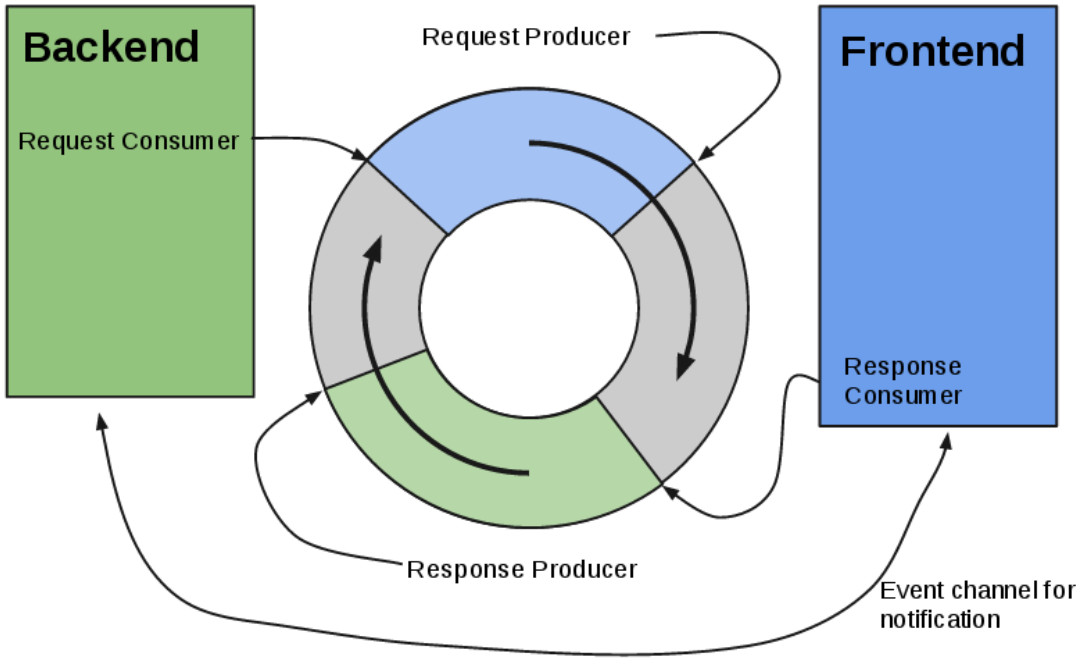
```
local = "" . . . . . . . . . . . . . . . . . . . . . . . . .  (n0)
 domain = "" . . . . . . . . . . . . . . . . . . . . . . . .  (n0)
  1 = "" . . . . . . . . . . . . . . . . . . . . . . . . . .  (n0,r1)
   name = "sle15sp4" . . . . . . . . . . . . . . . . . . . .  (n0,r1)
   cpu = ""  . . . . . . . . . . . . . . . . . . . . . . . .  (n0,r1)
    0 = "" . . . . . . . . . . . . . . . . . . . . . . . . .  (n0,r1)
     availability = "online" . . . . . . . . . . . . . . . .  (n0,r1)
    1 = "" . . . . . . . . . . . . . . . . . . . . . . . . .  (n0,r1)
     availability = "online" . . . . . . . . . . . . . . . .  (n0,r1)
   memory = "" . . . . . . . . . . . . . . . . . . . . . . .  (n0,r1)
    static-max = "4194304" . . . . . . . . . . . . . . . . .  (n0,r1)
    target = "4186112" . . . . . . . . . . . . . . . . . . .  (n0,r1)
    videoram = "8192"  . . . . . . . . . . . . . . . . . . .  (n0,r1)
   device = "" . . . . . . . . . . . . . . . . . . . . . . .  (n0,r1)
    vbd = "" . . . . . . . . . . . . . . . . . . . . . . . .  (n0,r1)
     768 = ""  . . . . . . . . . . . . . . . . . . . . . . .  (n1,r0)
      backend = "/local/domain/0/backend/vbd/1/768"  . . . .  (n1,r0)
      backend-id = "0" . . . . . . . . . . . . . . . . . . .  (n1,r0)
      state = "4"  . . . . . . . . . . . . . . . . . . . . .  (n1,r0)
      virtual-device = "768" . . . . . . . . . . . . . . . .  (n1,r0)
      device-type = "disk" . . . . . . . . . . . . . . . . .  (n1,r0)
      trusted = "1"  . . . . . . . . . . . . . . . . . . . .  (n1,r0)
      multi-queue-num-queues = "2" . . . . . . . . . . . . .  (n1,r0)
      queue-0 = "" . . . . . . . . . . . . . . . . . . . . .  (n1,r0)
       ring-ref = "8"  . . . . . . . . . . . . . . . . . . .  (n1,r0)
       event-channel = "16"  . . . . . . . . . . . . . . . .  (n1,r0)
      queue-1 = "" . . . . . . . . . . . . . . . . . . . . .  (n1,r0)
       ring-ref = "9"  . . . . . . . . . . . . . . . . . . .  (n1,r0)
       event-channel = "17"  . . . . . . . . . . . . . . . .  (n1,r0)
```

PV Geräte-Treiber

- Aktuell unterstützt: block, net, console, SCSI, USB, sound, display, 9pfs, pvcall, kbd, camera, framebuffer, TPM
- Gast enthält Frontend Treiber, Backend Treiber in dom0 oder Treiber-Domain
- Backend kann auch als User-Programm laufen (z.B. in qemu oder als daemon)
- Kommunikation geschieht über Ring-Puffer im Gastspeicher (von Backend über “*Grant*”-Mechanismus zugreifbar) und “*Events*” zur Signalisierung
- Ring-Puffer enthalten Aufträge und Antworten in Geräte-spezifischem Format
- Referenz auf I/O-Daten meißtens in Form von Puffern (*Grant*, Offset und Länge)

PV Geräte-Treiber

- Ring-Puffer enthält *Request* und *Response* Indizes für *Producer* und *Consumer*
- Jeder Auftrag und Antwort haben die gleiche Größe, der Ring-Puffer kann 2^n Aufträge bzw. Antworten aufnehmen
- Die andere Seite muss nur per Event Channel benachrichtigt werden, wenn der erste Auftrag bzw. die erste Antwort eingetragen wurde (*Consumer* war gleich *Producer*)



Virtio

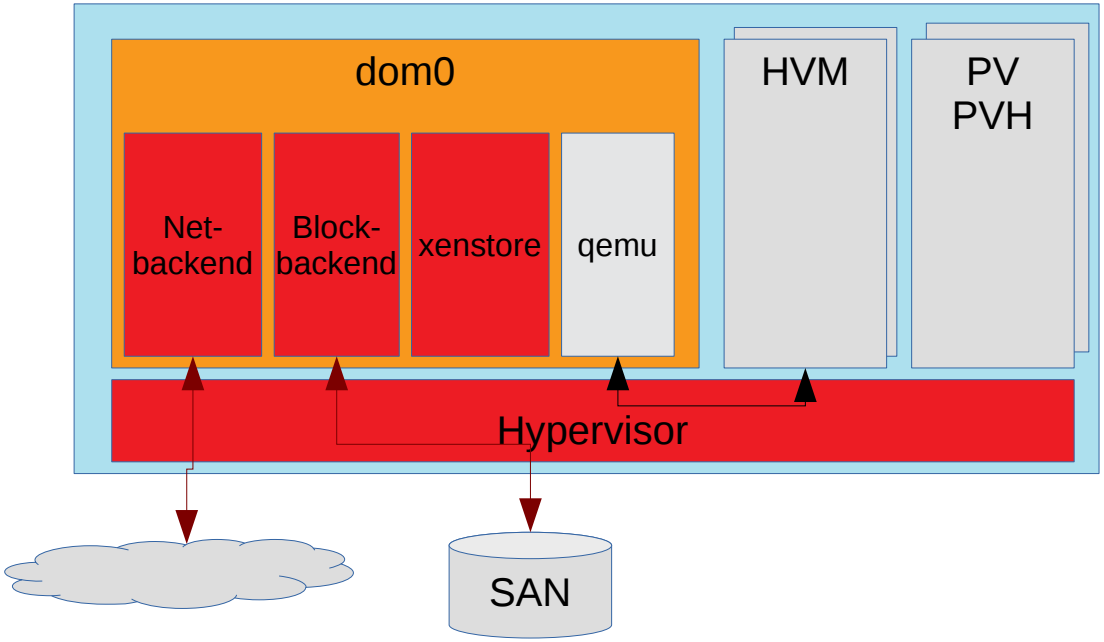
- Virtio unter Xen ist im Prinzip einfach möglich, wenn das Backend in dom0 läuft (nur dom0 darf beliebigen Speicher eines Gastes mappen)
- Seit kernel 5.19 beherrscht Linux auf Gastseite virtio mit Grants
- Qemu Patches für virtio Grant Support sind noch nicht integriert
- EPAM hat ein vhost-user Backend mit Grant Support für virtio Block-Devices geschrieben und setzt dieses in einer Treiber-Domain ein

Einsatzkonfigurationen von Xen

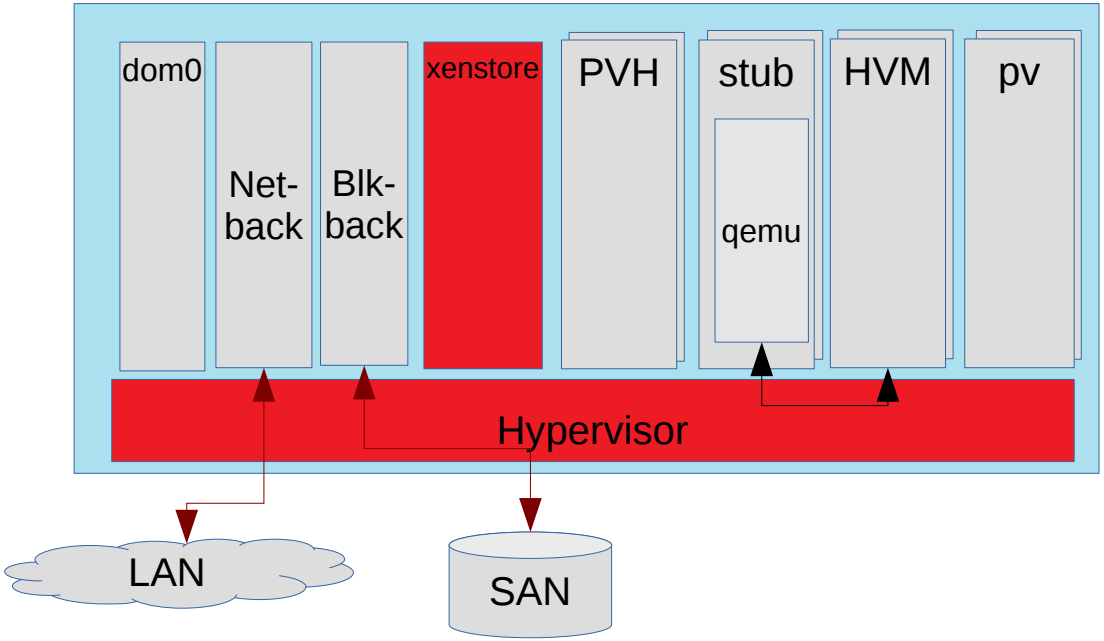
Einsatzkonfigurationen von Xen

- Standard (dom0 + domUs)
- “Disaggregated”
- “Dom0less”
- Mischformen

Standard (dom0 + domUs)



Disaggregated



Dom0less

- Spezialfall, bei dem der Hypervisor beim Bootvorgang bereits mehrere Gäste startet (zur Zeit nur für Arm unterstützt)
- Konfigurationsdaten der einzelnen Gäste (Speichergröße, Anzahl vcpus, Zugewiesene Geräte) werden dem Hypervisor im Device-Tree übermittelt
- Die Binaries (Kernel und Ramdisk) werden als Module in uboot spezifiziert
- Trotz des Namens “*Dom0less*” kann einer der Gäste durchaus als Dom0 fungieren

Mischformen

- Mischformen jeglicher Art denkbar
- z.B. Xenstore in eigener Domain (siehe openSUSE, SLE)
- Qubes-OS (qemu stubdom + Network Driver Domain + Desktop Driver Domain, ansonsten dom0)

Basismechanismen von Xen

Basismechanismen von Xen

- Scheduling
- Speicherverwaltung
- Hypercalls
- Grants
- Event Channels
- Device-Tree

Scheduling

- Der Scheduler des Xen Hypervisors ist verantwortlich für die Zuteilung von physikalischen CPUs an virtuelle CPUs der Gäste
- Welcher Scheduling-Algorythmus verwendet wird, ist einstellbar:
  - CREDIT oder CREDIT2 (allgemeine Einsatzzwecke, CREDIT2 ist moderner und optimiert für niedrige Latenz)
  - RTDS (Real-Time Scheduler)
  - ARINC653 (Real-Time mit fest zugeordneten Time-Slices)
  - NULL (Spezial Scheduler für weniger virtuelle als physikalische CPUs)

Scheduling

- Scheduler können Parameter haben (z.B. Time-Slice)
- Die Scheduling-Granularität (Thread, Core, Socket) kann eingestellt werden
- System kann in CPUPOOLS unterteilt werden:
  - Physikalische CPUs werden jeweils genau einem CPUPOOL zugeordnet, Zuordnung kann jedoch umkonfiguriert werden
  - Jeder Gast kann nur einen CPUPOOL nutzen, kann jedoch zwischen CPUPOOLs verschoben werden
  - Jeder CPUPOOL hat eigene Scheduling Parameter, es können also z.B. in einem CPUPOOL der CREDIT2 Scheduler verwendet werden und in einem anderen CPUPOOL der NULL Scheduler

Scheduling

- Für jede physikalische CPU gibt es immer eine virtuelle Idle CPU
- Virtuelle CPUs können an eine oder mehrere physikalische CPU gebunden werden (pinning), sowohl hart (dürfen nirgendwo sonst laufen), als auch weich (sollen möglichst dort laufen)
- Wann welche virtuelle CPU wo läuft, entscheidet der jeweilige Scheduler im Rahmen von Pinning und CPUPOOL (z.B. an Hand Priorität eines Gastes, in derVergangenheit verwendeter CPU-Zeit, …)
- Gründe für Scheduling sind Interrupts (Timer, I/O, Inter-Prozessor-Interrupts), Exceptions, oder Hypercalls

Speicherverwaltung

- Die Speicherverwaltung im Xen Hypervisor ist verantwortlich für:
  - Zuteilung von Speicher (Xen intern und an Gäste)
  - Verwaltung der Übersetzungstabellen
  - Isolation der Gäste bzgl. Speicherzugriffe
  - Konsistenz der Übersetzungstabellen (Cache-Kohärenz, Kohärenz der TLBs, etc.)
- Für einzelne Speicherseiten werden Referenzen gezählt, um Mehrfachverwendungen von Speicher sicher handhaben zu können

Hypercalls

- Hypercalls sind direkte Aufrufe des Hypervisors aus Gästen
- Der Hypervisor prüft bei jedem Hypercall, ob er für den aufrufenden Gast erlaubt ist
- Hypercalls, die nur von Xen Tools genutzt werden sollten, haben kein stabiles ABI (sysctl und domctl Hypercalls)
- Für bessere Performance können mehrere Hypercalls als “Batch” zusammengefasst werden (“Multicall” Hypercall)
- Parameter werden in Registern übergeben, diese können auch Gastspeicherbereiche adressieren
- Hypercalls sind nur aus privilegiertem Code erlaubt, Xen Tools nutzen daher einen Passthrough Treiber im Kernel (xen-privcmd).

Grants

- Grants dienem einem Gast, um anderen Gästen Zugriff auf den eigenen Speicher zu erlauben (z.B. einem Backendtreiber zur Durchführung einer I/O)
- Grants werden in Grant-Pages verwaltet, die der Gast dem Hypervisor bekannt gibt
- Referenziert werden Grants über den jeweiligen Grant-Index (zusammengesetzt aus dem Index der Grant-Page und dem Index innerhalb der Grant-Page)
- Jeder Grant enthält die Domain-Id des Gastes, der zugreifen darf, die Art des erlaubten Zugriffs (R/W), die Seitennummer auf die zugegriffen werden darf und Status-Daten des Hypervisors (z.B. Grant ist in Verwendung durch anderen Gast)

Grants

- Verwendung von Grants geschieht entweder durch “map” oder durch “copy” Operationen per Hypercall
- Während ein Grant in Verwendung ist, kann er nicht aufgehoben werden (Mapping kann nur freiwillig aufgegeben werden)
- Sowohl Anzahl der Grant-Pages als auch Anzahl der gleichzeitig erlaubten Grant-Mappings ist pro Gast konfigurierbar (Grund: es werden entsprechend große Verwaltungs-Datenstrukturen im Hypervisor benötigt)
- Es gibt aktuell 2 Grant-Formate (V1: 8 Bytes pro Grant, V2: 16 Bytes pro Grant); V2 wird bei mehr als 16TB Speicher benötigt
- In Linux existiert eine Bibliothek zur Nutzung in User-Programmen

Event Channels

- Event Channels sind ein Xen-spezifischer Interrupt-Ersatz
- Sie können zwischen 2 Gästen (bidirektional), innerhalb eines Gastes, oder zwischen Gast und Hypervisor genutzt werden
- 2 Formate: 2-Level (bis zu 4096 Event Channels pro Gast), oder FIFO (bis zu 131072 Event Channels pro Gast), Format kann vom Gast gewählt werden
- In Linux existiert eine Bibliothek zur Nutzung in User-Programmen

Device-Tree

- Nur relevant für Arm
- Dokumentation unter `docs/misc/arm/device-tree` in Xen Sources
- Für System-Boot zusätzliche Einträge für Xen unter `/chosen`
- Für domU Boot Einträge unter `/hypervisor`

Device-Tree System-Boot

- Enthält Konfigurationsdetails für dom0 und ggf. domUs:
  - Ladeadressen für Kernel und Ramdisk
  - Xen Feature Initialisierung (optional, z.B. Events, Grants, Xenstore)
  - Shared Memory Bereiche (optional)
- Optional können Cpupools konfiguriert werden
- Geräte mit der Property `“xen,passthrough”` werden nicht an dom0 gegeben (Passthrough)

Device-Tree domU-Boot

- Enthält:
  - Spezifikation der Xen-Version (für Ermittlung der verfügbaren Interfaces)
  - Adressbereich für Grant-Table
  - Adressbereiche für Grant-Mappings (optional)
  - Interrupt für Events
  - Definitionen für Unterstützung von UEFI (optional)
- Zusätzlich: unter `/chosen` Eintrag für ACPI-Daten

Konfigurieren und Bauen von Xen

Konfigurieren und Bauen von Xen

- Xen Sources
- Build Abhängigkeiten
- Konfiguration des Builds
- Konfiguration des Hypervisors
- Eigentlicher Build

Xen Sources

- Xen Upstream Sources unter **git://xenbits.xen.org/xen.git**
- Relevante Verzeichnisse:
  - xen – Hypervisor
  - tools – Xen tools (alles, was in Xen-System im User-mode läuft)
  - docs – Dokumentation
  - stubdom – diverse Stub-domain Sources (nur auf x86 relevant)
  - Der Rest ist relevant für das Bauen an sich oder für Test-Automatisierung

Build Abhängigkeiten

- Vorausgesetzte Tools (Liste unvollständig): make, gcc, binutils, awk, bash, python, …
- Je nach Konfiguration des Builds können weitere Sources aus anderen Repositories während des Build-Prozesses nachgeladen werden (vor Allem bei x86, wie z.B. qemu, OVMF, Seabios, grub, …)
- Erforderliche Komponenten (Bibliotheken, Header) werden bei der Konfiguration des Builds fast alle auf Verfügbarkeit getestet und entsprechend mit Fehler abgebrochen, falls sie fehlen
- Optionale Komponenten, die nicht verfügbar sind, führen teilweise zu einer stillen Deaktivierung von Features

Konfiguration des Builds

- Als erster Schritt beim Bauen von Xen muss der Build mittels `./configure` konfiguriert werden
- Mit `./configure --help` können die wesentlichen Optionen angezeigt werden
- Configure muss nur erneut verwendet werden, wenn Optionen geändert werden sollen, oder wenn größere Updates der Xen Sources durchgeführt wurden (z.B. neue Xen Version)
- Es ist sinnvoll, die Ausgaben von configure genau zu prüfen, falls wegen fehlender Header/Bibliotheken ein Feature still deaktiviert wurde

Konfiguration des Hypervisors

- Der Hypervisor lässt sich wie der Linux Kernel bzgl. verfügbarer Komponenten konfigurieren
- Konfigurationssystem ist *Kconfig* wie beim Linux Kernel
- Aufruf z.B. mittels `make -C xen menuconfig`
- Konfiguration wird in xen/.config abgelegt
- Konfiguration des laufenden Hypervisors kann aus dom0 mit dem Aufruf von `xenhypfs cat /buildinfo/config` ermittelt werden

Eigentlicher Build

- Build kann z.B. mit `make -j 8` gestartet werden
- Installation von Xen wie konfiguriert auf der lokalen Anlage mit `make install`
- Mit `make help` können weitere make Optionen angezeigt werden

Abläufe im Detail

Abläufe im Detail

- Bootvorgang von Xen
- Start eines Gastes
- I/O eines Gastes mit PV-Treibern
- PCI Passthrough

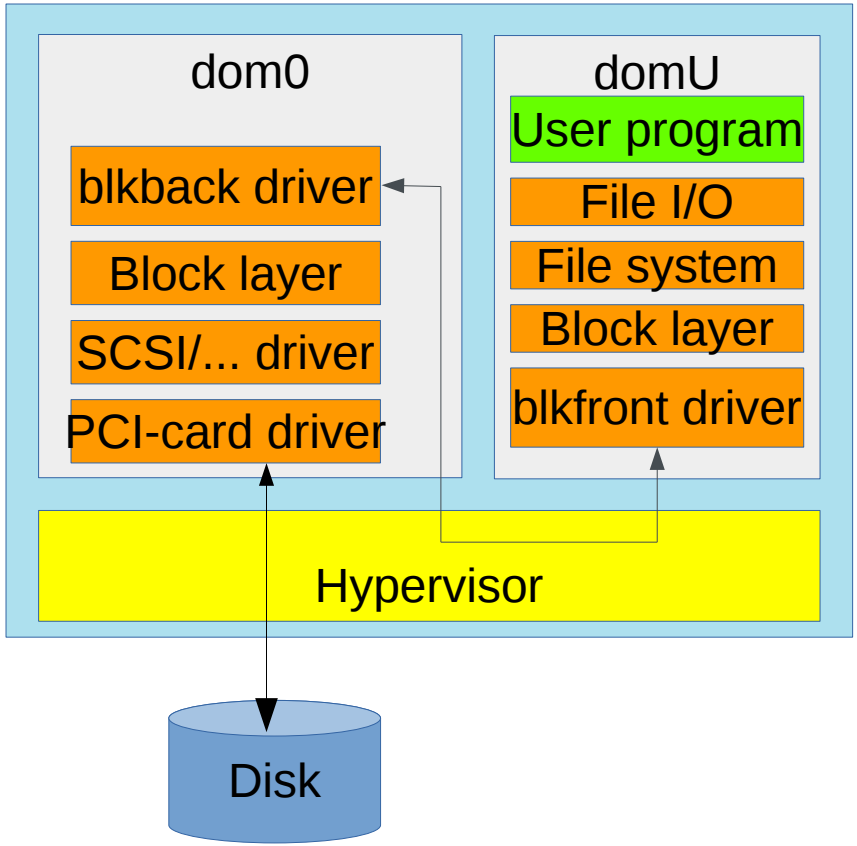
Bootvorgang von Xen

- Bootloader lädt Xen Hypervisor, dom0 Kernel und dom0 Ramdisk (auf Arm: auch Device-Tree und evtl. weitere Kernels und Ramdisks) in den Speicher und ruft den Hypervisor auf
- Hypervisor initialisiert sich und aktiviert virtuelle Adressierung
- Hypervisor analysiert Hardware und initialisiert diese (Start anderer Prozessoren, Ermittlung der Speichergröße, verügbare Features)
- Hypervisor initialisiert Konfiguration von dom0 (Speichergröße, etc.) und aktiviert dom0
- Dom0 Bootet virtualisiert
- Dom0 erkennt, dass unter Xen als dom0 gestartet wurde, und initialisiert Xen-spezifische Treiber und Prozesse

Start eines Gastes

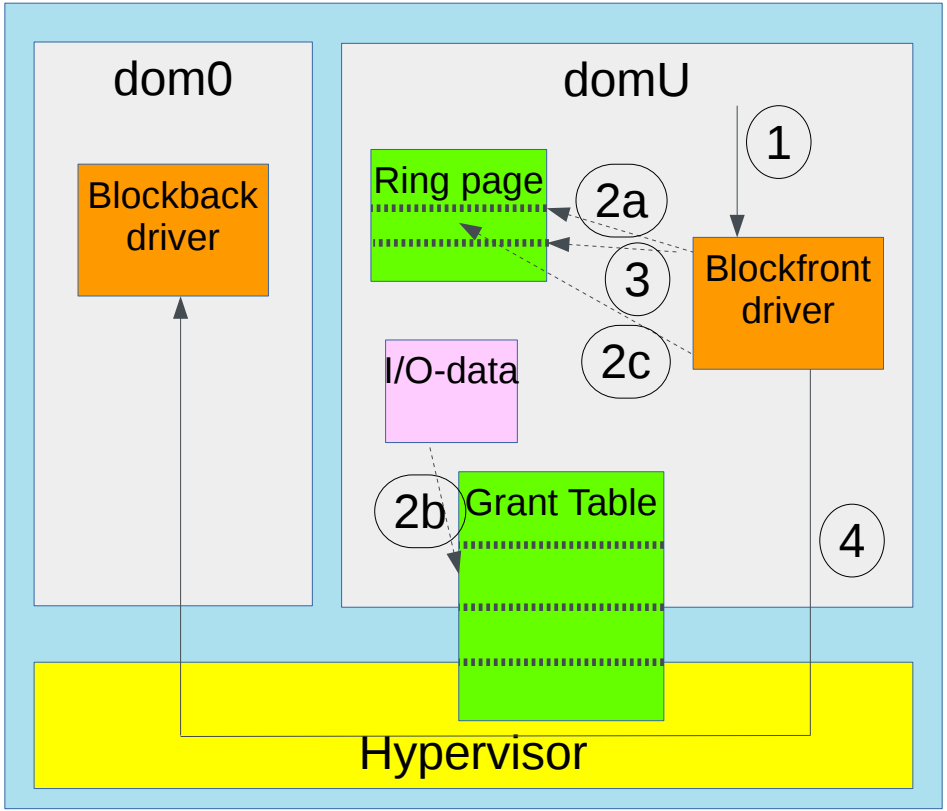
- Xen tools (xl/libxl oder libvirt/libxl) analysieren Konfiguration des Gastes
- Aufruf des Hypervisors zum kreieren eines neuen Gastes mit den wichtigsten Parametern (Gasttyp, Anzahl der Prozessoren, max. Größe des Speichers, weitere Ressource-Limits, Scheduling-Parameter)
- Allokierung des initialen Speichers für den Gast über den Hypervisor
- Konfiguration der Geräte für den Gast (Schreiben der Xenstore-Einträge, evtl. Starten von Backends im User-Mode)
- Initialisierung der virtuellen Boot-CPU des Gastes
- Aktivierung der virtuellen Boot-CPU des Gastes
- Gast startet, erkennt Xen, initialisiert Xen-spezifische Treiber (Events, Grants, Xenstore) und liest aus dem Xenstore die Gerätekonfiguration und initialisiert die Geräte

I/O eines Gastes mit PV-Treibern



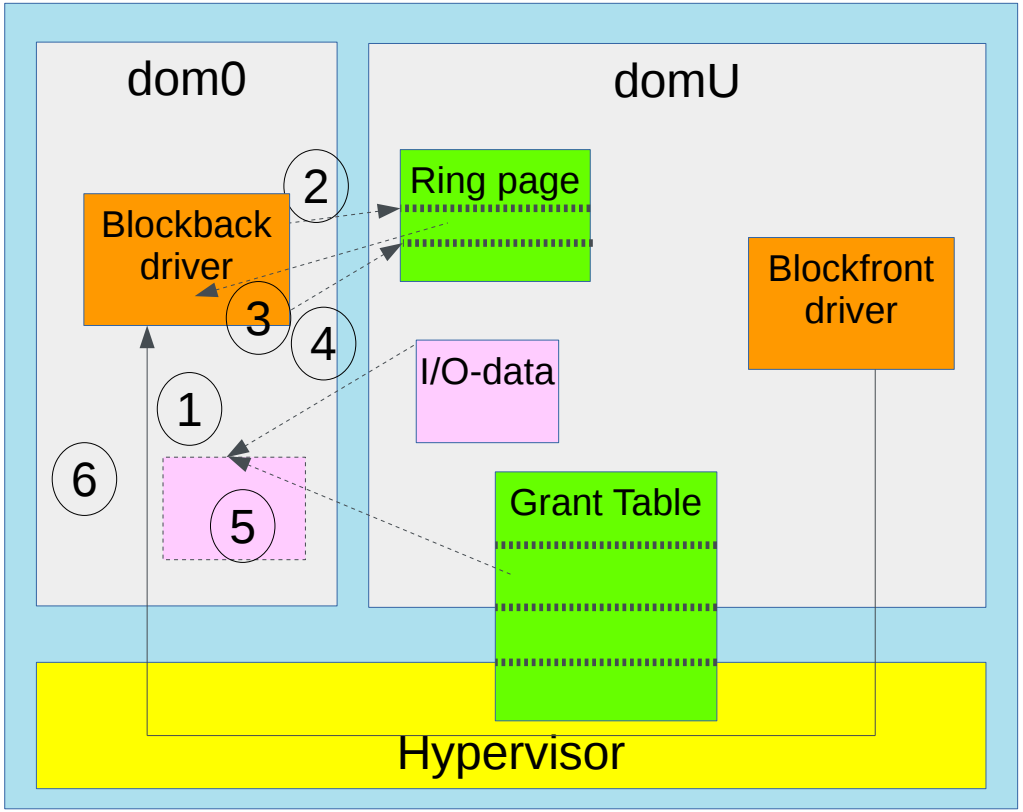
I/O eines Gastes mit PV-Treibern

1) I/O vom Benutzer gestartet, erreicht Frontend
2) Auftrag für Backend:
   a) Request Slot in Ring-Page allokiert
   b) Grants für I/O-Daten initialisiert
   c) Request Slot mit Daten gefüllt
3) Request Producer Index erhöht
4) Event an Backend schicken



I/O eines Gastes mit PV-Treibern

1) Event erreicht Backend
2) Backend findet neuen Auftrag in Ring-Page
3) Auftragskopie in lokalem Speicher angefertigt (vermeidet Änderung durch Frontend während Verarbeitung)
4) Request Consumer Index erhöht
5) I/O-Daten über Grants in Speicher gemappt
6) I/O wird lokal gestartet



I/O eines Gastes mit PV-Treibern

- I/O-Ende im Backend:
  - I/O-Ende-Funktion im Backend wird aktiviert
  - Backend allokiert Response Slot in der Ring-Page
  - Response Slot wird mit I/O-Ende Daten gefüllt
  - Grant Mappings werden freigegeben
  - Response Producer Index wird erhöht
  - Event wird an Frontend gesendet
- I/O-Ende im Frontend:
  - Event wird empfangen
  - Frontend findet neue Antwort in der Ring-Page
  - Grants für I/O-Seiten werden entfernt
  - Response Consumer Index wird erhöht
  - I/O-Ende-Funktion des Aufrufers wird gerufen

PCI Passthrough

- Ein komplettes PCI-Gerät wird einem Gast zugewiesen
- Für sicheren Betrieb ist eine IOMMU erforderlich (verhindert DMA auf Speicher außerhalb des Gastes)
- Bei PV-Gästen geschehen Zugriffe auf den Config-Bereich des PCI-Gerätes über den xen-pcifront Treiber, der mit xen-pciback in Dom0 kommuniziert (manche Operationen erfordern privilegierte Befehle, die in PV-Gästen nicht verfügbar sind)
- PCI-Geräte können im laufenden Betrieb Gästen zugewiesen und wieder entfernt werden

Debugging von Xen

Debugging von Xen

- Hypervisor Debugging allgemein
- Xentrace
- DEBUG Optionen beim Bauen
- Debugging über serielle Schnittstelle
- Debugging einer DomU aus Dom0
- Logfiles in Xen

Hypervisor Debugging allgemein

- Hypervisor Debugging möglich über:
  - Konsolausgaben
  - Xentrace
  - Kdump (erfordert Support in dom0)
- Konsolausgaben vom Hypervisor sichtbar über Konsole, serielles Interface oder `xl dmesg`
- Manche Debughilfen stehen nur zur Verfügung, wenn der Hypervisor mit speziellen DEBUG Optionen gebaut wurde
- Über Boot-Parameter (siehe `docs/misc/xen-command-line.pandoc`) lassen sich Konsol-Ausgaben teilweise einstellen

Xentrace

- Benötigt CONFIG_TRACEBUFFER (normalerweise aktiv) um zu funktionieren
- Schreibt Trace-Einträge (Hypercalls, Scheduling Infos, ..) in Tracepuffer zur späteren Auswertung
- Kontrolle über `xentrace`, Auswertung über `xenalyze`
- Beispiel siehe Live Vorführung

DEBUG Optionen beim Bauen

- Definiert in xen/Kconfig.debug, erfordern CONFIG_DEBUG oder CONFIG_EXPERT zur Aktivierung
- Zu unterscheiden sind verschiedene Gruppen:
  - Aktivierung von zusätzlichen Debugging Funktionen (DEBUG_LOCK_PROFILE, PERF_COUNTERS, DEBUG_TRACE, ...)
  - Aktivierung von Konsistenzprüfungen (DEBUG_LOCKS, UBSAN, ...)
  - Erweiterte Debug Daten (DEBUG_INFO, VERBOSE_DEBUG, FRAME_POINTER, …)
- Allgemein erhöhter Speicherbedarf und u.U. schlechtere Performance

Debugging über serielle Schnittstelle

- Input über serielle Schnittstelle kann mit 3-maligem ctrl-A zwischen Hypervisor und dom0 umgeschaltet werden
- Verschiedene “debug-keys” lösen diverse Diagnoseausgaben aus (“h” liefert Liste der möglichen Tasten)
- Ermöglicht rudimentäre Diagnose eines Systems mit hängender dom0
- Mit `xl debug-key <taste>` in dom0 kann die Ausgabe auch angestoßen werden

Debugging einer DomU aus Dom0

- Eine domU kann mittels `xl pause` und `xl unpause` angehalten bzw. wieder angestartet werden
- Mit dem Programm `/usr/lib/xen/bin/xenctx` können Register und Speicherinhalte der domU angezeigt werden
- Mit `xenstore-ls /local/domain/<domid>` können Xenstore-Inhalte einer domU angezeigt werden

Logfiles in Xen

- Xenstore Logging unter /var/log/xen/xenstore (kontrollierbar mit dem Programm `xenstore-control` oder mit entsprechenden Parametern beim Start von Xenstore)
- Falls aktiviert werden domU-Konsol logs unter /var/log/xen/console abgelegt
- Logs von xl unter /var/log/xen/xl-<domain-name>.log* (normalerweise die letzten 11 Instanzen)
- Auf x86 außerdem noch qemu-<domain-name>.log*

Live Vorführung

Live Vorführung

- Gast starten
- Xen bauen
- Debugging

Zusätzliche Informationen

Zusätzliche Informationen

- Upstream Xen
- Xilinx und Zephyr

Upstream Xen

- https://wiki.xenproject.org/wiki/Main_Page
- https://wiki.xenproject.org/wiki/Xen_Man_Pages
- Im Source-Tree unter docs/ (z.B. docs/features/dom0less.pandoc)
- Hypervisor Interfaces im Source-Tree unter xen/include/public

Xilinx und Zephyr

Auszug aus einer Mail von einem Xilinx Xen Entwickler:

- Downstream Xen
Latest version publicly available (soon there will be xlnx_rebase_4.18):
https://github.com/Xilinx/xen/tree/xlnx_rebase_4.17
The are some differences between a downstream fork and upstream Xen like EEMI mediator
for firmware related stuff (e.g. power/clock management on Xilinx boards), cache coloring,
virtio-net, PCI passthrough and more.
- Hardware
ZCU102 is a Xilinx UltraScale+ MPSoC dev board. It has 4 x Cortex-A53 + 2 x Cortex-R5,
SMMUv2, GICv2 and tons of other peripherals that are not to be used by Xen itself (e.g. GEM, TTC,
SATA, USB, etc.).
- Building
All the images including firmware, bootloader, Xen, Linux can be build using Petalinux (a wrapper around Yocto stuff)
or Yocto.
Instructions for Petalinux:
https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/2709946369/Building+Xen+Hypervisor
Instructions for Yocto:
https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/1696137838/Building+Xen+Hypervisor+through+Yocto+Flow
The machine name for ZCU102 is:
MACHINE=zcu102-zynqmp
- Zephyr with dom0less Xen
Running Zephyr as a dom0less domU is possible and tested. There is nothing AMD/Xilinx specific here. Please refer to:
https://docs.zephyrproject.org/latest/boards/xen/xenvm/doc/index.html
- Xen with Linux and Zephyr
Presentation from Stefano Stabellini:
https://www.slideshare.net/StefanoStabellini/static-partitioning-with-xen-linuxrt-and-zephyr-a-concrete-endtoend-example-elc-na-2022
- Other information
There are bunch of other useful pages under:
https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842530/Xen+Hypervisor
ZCU102 has been the most popular and best tested Xilinx board for years using Xen.