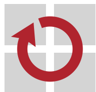
Praktikum angewandte Systemsoftwaretechnik (PASST)

Arbeitsumgebung / Aufgabe 1

2. Mai 2019

Tobias Langer, Stefan Reif, Michael Eischer
und Florian Schmaus

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg



Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Rechnerzugang

- Rechnerarbeit im ehem. *WinCIP* (Raum: **01.153-113**).
- Rechner: `faui01[a-r]`, `faui09[a-j]`
- Reguläre CIP Accounts
- Zusätzlich:
  - Testrechner nach Bedarf

Arbeitsumgebung im CIP

Arbeiten im CIP

- Begrenztes Quota
  - Größere Installationen daher `/proj/ciptmp`
  - Falls noch nicht geschehen: Verzeichnis anlegen
  - Vorsicht: Für `/proj/ciptmp` gibt es kein Backup
- `kvm` über SSH nicht möglich
  - ⇒ Erfordert lokales Arbeiten
- PASST Quellen sind unter `/proj/i4passt` eingeblendet

Debian in eine KVM installieren

Virtualisierungswerkzeug: KVM

- Entwickeln und Testen mit virtuellen Maschinen
  - + Schnellere Umlaufzeiten
  - + Erleichtert Debugging
  - – Nicht 100% exaktes Systemverhalten

    ...für unsere Ansprüche gut genug.
- Empfehlung: QEMU/KVM
  - Grundsätzlich sind auch Virtualbox oder VMware möglich.

Virtuelle Festplatte vorbereiten

```
$ dd if=/dev/zero of=passt.img bs=1 count=1 seek=8G
$ du -sh passt.img
4,0K    passt.img
```

Virtuelle Festplatte anlegen

- Erstellt eine **virtuelle** Festplatte in der Datei `passt.img`
- Nicht allokierter Platz wird auch tatsächlich nicht belegt (**sparse file**)
- Mit `qemu-img(8)` können auch Abbilder in besseren Formaten (z.B. `qcow2`) angelegt werden.

Starten der virtuellen Maschine mit KVM

```
kvm -m 1024 -nodefaults -nographic -display none \
-echr 0x01 -serial mon:stdio \
-serial tcp::9876,server,nowait,nodelay \
-net nic,model=virtio -net user \
-drive file=passt.img,if=virtio,cache=writeback,format=raw \
<kernel_binary>
```

- `-nographic` deaktiviert Grafikkartenemulation
  - Interaktion über serielle Konsole (in Linux `/dev/ttyS0`)
- Fallstricke:
  - Netzwerk wird per NAT zur Verfügung gestellt → ICMP (also z.B. `ping`) funktioniert nicht.

Escape Key

Im QEMU/KVM ist der Escape Key auf Ctrl + a aktiviert.
Folgende Kommandos sind dann verfügbar:

h	Hilfe anzeigen
x	Emulator beenden
s	Festplattendaten in Datei speichern (bei `-snapshot`)
t	Konsolenzeitstempel umschalten
b	Abbruch senden (Magic SysRq)
c	Zwischen Konsole und Monitor umschalten
Ctrl + a	Sende Ctrl + a in die VM

Ein **emergency sync** (SysRq) kann damit so ausgelöst werden:
Ctrl + a, b, s.

Minimales Installationsprogram laden

```
host="http://ftp.fau.de/\
debian/dists/stretch/main/installer-amd64/\
current/images/netboot/debian-installer/amd64/"
wget $host/linux
wget $host/initrd.gz
```

Download Debian netinst Installer

- Minimaler Debian Installer (nur wenige MiB gross)
  - Kernel + Ramdisk (Installationsprogramm).
- Alles Weitere wird vom Debian Spiegel nachgeladen.
- Für die eigentliche Installation den Spiegel `ftp.fau.de` benutzen!

Installation im Textmodus

```
boot.sh -kernel linux -initrd initrd.gz \
-append "console=ttyS0 priority=low"
Loading Linux 2.6.32.38 ...
Loading initial ramdisk ...
[    0.000000] Initializing cgroup subsys cpuset
[    0.000000] Initializing cgroup subsys cpu
[    0.000000] Linux version 2.6.32.38 (root@faui48d)
```

Linux Kern übersetzen

Quellen vorbereiten

```
$ git clone /proj/i4passt/kernel/linux-stable.git
git clone /proj/i4passt/kernel/linux-stable.git
Cloning into linux-stable...
done.
```

Klonen der Linux Quellen von i4 Quellen

- Konfiguration von Linux mittels

  ```
  make menuconfig
  ```

  oder

  ```
  make xconfig
  ```

- Bauen mittels

  ```
  make -j 4
  ```

Kernel-Optionen

- Nützliche Optionen

  `CONFIG_64BIT` 64-Bit Kernel

  VIRTIO-Treiber Paravirtualisierte Treiber

- Module gegebenenfalls statisch binden.
  - Erspart das Erstellen der Initramfs
  - Kein manuelles Laden der Module in GDB.

  → Module in der Kconfig ausschalten

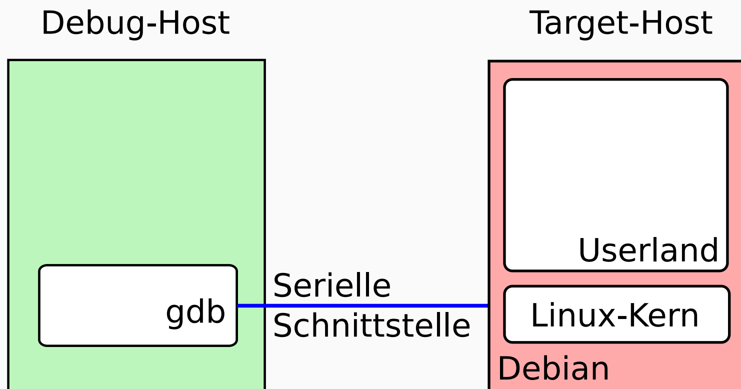
- Suche in `make menuconfig`: / tippen
  - Ziffer springt zum jeweiligen Ergebnis

- Nutzlose Treiber rauswerfen: Firewire, Sound, Multimedia, obskure Netzwerkprotokolle...

Aufgabe: Wer schafft den kleinsten Kernel?

Kernel Debugger konfigurieren

Logisches Rechnersetup



Wichtige Debug-Kernel-Optionen

`CONFIG_DEBUG_INFO`
: Übersetzt den Kernel mit Debuginformationen.

`CONFIG_FRAME_POINTER`
: Unterbindet das Wegoptimieren des Framepointers.

`CONFIG_GDB_SCRIPTS`
: Aktiviert GDB-Skripte zum leichteren Kernel-Debugging.

`RANDOMIZE_BASE`, `RANDOMIZE_MEMORY`
: ASLR, randomisiert Speicheraddressen, verwirrt GDB

Diese Liste ist nicht vollständig…

Booten des Kerns mittels QEMU

- QEMU implementiert eigenen Bootloader
- Über Kommandozeile werden die Bootparameter übergeben:

  `-kernel` Pfad zu bzImage

  `-append` Kernelparameter (durch Leerzeichen getrennt, s.u.)

  `-initrd` Bei Bedarf: Pfad zur Initramfs

- Nützliche Kernelparameter

  `kgdboc=ttyS1,115200` KGDB konfigurieren

  `kgdbwait` Beim Booten auf eine GDB-Verbindung warten

  `root=/dev/vda1` Root-Dateisystem auf virtio-Platte

Alternative: Installation des Kernels in die VM

1. Auf dem Buildhost:

   ```
   fakeroot make deb-pkg V=1 -j4
   ```

2. `.deb` Dateien in VM mit `dpkg -i *.deb` installieren
3. Geeignete Kernel-Boot-Optionen setzen!

   ```
   GRUB_DEFAULT=0
   GRUB_TIMEOUT=5
   GRUB_CMDLINE_LINUX_DEFAULT="verbose"
   GRUB_CMDLINE_LINUX="console=ttyS0"\
                      "kgdboc=ttyS1,115200 root=/dev/vda1"
   GRUB_TERMINAL=serial
   GRUB_SERIAL_COMMAND="serial --unit=0 --speed=115200 --stop=1"
   ```

4. Aktivieren der Änderungen: `update-grub`
5. Neustarten: `reboot`

Umgang mit dem Kernel Debugger

Debuggen mit dem GDB

- Programm muss mit Debugsymbolen (`-g`) übersetzt werden.
  In Linux gibt es hierfür eine Konfigurationsoption.
- Normalerweise (wie z.B. in Systemprogrammierung) werden *lokale* Anwendungen untersucht.
- In PASST: *remote debugging*.
- Unterbrechen funktioniert nicht via kgdb.
- Stattdessen aus der VM:

```
echo g >/proc/sysrq-trigger
```

Debuggen mit dem GDB

```
$  gdb vmlinux
[...]
Reading symbols from /build/foo/linux-2.6.38/vmlinux
...done.

  (gdb) target remote localhost:9876

Remote debugging using localhost:9876
kgdb_breakpoint (new_dbg_io_ops=<value optimized out>)
at /build/foo/linux-2.6.38/kernel/debug/debug_core.c:960
960      wmb(); /* Sync point after breakpoint */

  (gdb)
```

Breakpoints

- Anlegen mit `(b)reak`

  `b [<Dateiname>:]<Funktionsname>`

  `b <Dateiname>:<Zeilennummer>`

  `b *<Adresse>`

  Breakpoint im open-Systemaufruf: `b __x64_sys_open`
- Fortfahren der Ausführung mit `(c)ontinue`
- Schrittweise Abarbeitung auf Ebene der Quellsprache mit

  `(s)tep` läuft in Funktionen hinein

  `(n)ext` behandelt Funktionsaufrufe als einzelne Anweisung

  `finish` läuft bis zum Ende der aktuellen Funktion
- Breakpoints anzeigen: `info breakpoints`
- Breakpoint löschen: `delete breakpoint`

Variablen, Stack

`(p)rint expr` Anzeigen von Variablen (`expr` kann auch C-Ausdruck sein)

`display expr` Anzeige von Variablen bei jedem Programmstopp (Breakpoint, Step, ...)

`set <variablenname>=<wert>` Setzen von Variablenwerten

`bt` (backtrace) Ausgabe des Aufrufstacks

Watchpoints

Stoppen Ausführung bei Zugriff auf eine bestimmte Variable

`watch expr`	Stoppt, wenn sich der Wert des C-Ausdrucks `expr` ändert
`rwatch expr`	Stoppt, wenn `expr` gelesen wird
`awatch expr`	Stopp bei jedem Zugriff (kombiniert `watch` und `rwatch`)

Anzeigen und Löschen analog zu den Breakpoints

Dokumentation

Documentation/dev-tools/gdb-kernel-debugging.rst

Fragen?