

Praktikum angewandte Systemsoftwaretechnik (PASST)

Arbeitsumgebung / Aufgabe 1

25. Oktober 2018

Tobias Langer, Stefan Reif, Michael Eischer,
Bernhard Heinloth 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

- Die Rechnerarbeit findet in der *Manlobbi* (Raum: **0.058-113**) statt.
- Es sind spezielle Zugänge erforderlich
- Zusätzliche Software kann auf Anfrage problemlos nachinstalliert werden.
- Es wird nur /home und /proj gesichert.

Neue (bevorzugte) Termine nach Umfrage

- Dienstag, 10 – 12 Uhr
- Donnerstag, 10 – 12 Uhr

- Rechner: faui49man[1-12]
- Logins: Erstellen gleich in Rechnerübung
- /home mit Backup aber begrenzte Größe und NFS
- lokal für große Dinge und wenns schnell gehen soll:
 /srv/scratch/\$USER
 (z.B. Kernel bauen)
- notfalls auch per NFS über
 /net/faui49man11/srv/scratch/...

Debian in eine KVM installieren

Virtualisierungswerkzeug: KVM

- In PASST entwickeln und arbeiten wir mit Virtualisierungstechniken.
- Wir empfehlen QEMU/KVM, Bearbeitung ist aber auch z.B. mit Virtualbox oder VMware möglich.
- Für KVM (Hardwaresupport) werden Schreibrechte auf /dev/kvm benötigt:

Zugriffsrechte bzw. ACLs auf /dev/kvm prüfen

```
01 $ ls -la /dev/kvm
02 crw-rw-rw-+ 1 root kvm 10, 332 2012-04-29 21:43 /dev/kvm
```

Virtuelle Festplatte anlegen

```
01 $ dd if=/dev/zero of=p_passt.img bs=1 count=1 seek=8G
02 $ du -sh p_passt.img
03 4,0K          p_passt.img
```

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

Hilfscript beim Umgang mit KVM

Datei /proj/i4passt/boot.sh

```
01 #!/bin/sh
02 kvm -m 1024 -nodefaults -nographic \
03   -chr 0x01 -serial mon:stdio \
04   -serial tcp:localhost:`id -u`,server,nowait,nodelay \
05   -net nic,model=virtio -net user \
06   -drive file=p_passt.img,if=virtio,cache=writeback \
07   "$@"
```

- Häufig benutzte Optionen werden mit diesem Skript gekapselt; weitere Optionen können angehängt werden
- Mittels `-nographic` wird die Graphikkarte nicht simuliert, d.h. Interaktion mit System ist nur via serieller Konsole (unter Linux `/dev/ttyS0`) möglich!
- Es wird ein Netzwerk 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 Installationsprogramm laden

Download Debian Installer (netinst)

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

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

Installation im Textmodus

Starten mittels Hilfsscript

```
01 /proj/i4passt/boot.sh -kernel linux -initrd initrd.gz \  
02     -append "console=ttyS0 priority=low"  
03 Loading Linux 2.6.32.38 ...  
04 Loading initial ramdisk ...  
05 [    0.000000] Initializing cgroup subsys cpuset  
06 [    0.000000] Initializing cgroup subsys cpu  
07 [    0.000000] Linux version 2.6.32.38 (root@fau148d)
```

Linux Kern übersetzen

- Im Labornetz existiert ein Spiegel des Linux-Kernels

Klonen der Linux Quellen

```
01 $ mkdir /srv/scratch/$USER/  
02 $ cd /srv/scratch/$USER/  
03 $ git clone /proj/i4passt/kernel/linux-stable.git  
04 git clone /proj/i4passt/kernel/linux-stable.git  
05 Cloning into linux-stable...  
06 done.
```

- Konfiguration von Linux mittels

```
make menuconfig
```

oder

```
make xconfig
```

- Bauen mittels

```
make -j 4
```

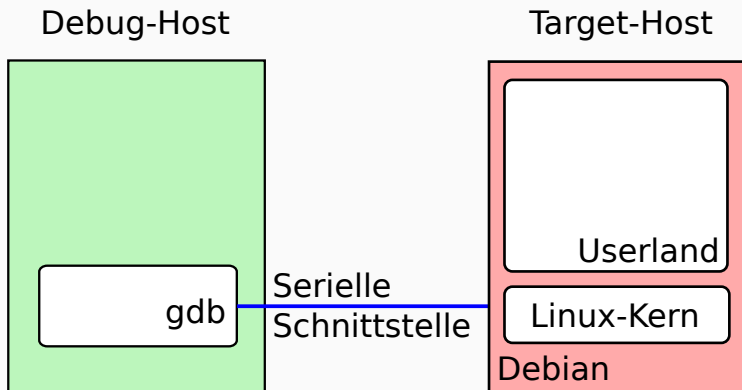
Kernel-Optionen

- **CONFIG_VIRTIO_BLK, CONFIG_SCSI_VIRTIO**
 - Paravirtualisierte Plattentreiber. Schneller und default bei Debian-Installation.
 - restliche VIRTIO-Optionen oft auch sinnvoll
- **CONFIG_64BIT**
- Module gegebenenfalls statisch binden.
 - Erspart das Erstellen der Initramfs (erleichtert möglicherweise das Laden des Kernels durch QEMU)
 - Module müssen nicht manuell in GDB geladen werden.
 - Module in der Kconfig ausschalten → alle Module sind statisch.
- Suche in make menuconfig: tippen
 - Ziffer springt zum jeweiligen Ergebnis
- Nutzlose Treiber rauswerfen: Firewire, Sound, Multimedia, obskure Netzwerkprotokolle...
 - Beschleunigt make und reduziert Fehlerquellen.

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 Speicheradressen, verwirrt GDB

Diese Liste ist nicht vollständig...

Booten des Kerns mittels QEMU

- QEMU implementiert eigenen Bootloader, so dass der Bootloader nicht unbedingt nötig ist.
- 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 (default bei VMs mit Debian, statt früher /dev/sda1)

Alternative: Installation des Kernels in die VM

- Auf dem Buildhost:
 `fakeroot make deb-pkg V=1 -j4`
- Entstehende `.deb` Dateien per `scp` in die virtuelle Maschine kopieren, und mit `dpkg -i *.deb` installieren
- Geeignete Kernel-Boot-Optionen setzen!

Grub-Optionen in `/etc/default/grub`

```
01 GRUB_DEFAULT=0
02 GRUB_TIMEOUT=5
03 GRUB_CMDLINE_LINUX_DEFAULT="verbose"
04 GRUB_CMDLINE_LINUX="console=ttyS0 kgdboc=ttyS1
    ↪ ,115200 root=/dev/vda1"
05 GRUB_TERMINAL=serial
06 GRUB_SERIAL_COMMAND="serial --unit=0 --speed=115200
    ↪ --stop=1"
```

- Aktivieren der Änderungen: `update-grub`
- Neustarten: `reboot`

Umgang mit dem Kernel Debugger

Debuggen mit dem GDB

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

Der laufende Linux-Kernel kann so unterbrochen werden

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

Debuggen mit dem GDB

Aufruf und Verbindung zum entfernten Kern:

```
01 $ gdb vmlinux
02 [...]
03 Reading symbols from /build/foo/linux-2.6.38/vmlinux
04 ...done.
05
06 (gdb) target remote localhost:4444
07
08 Remote debugging using localhost:4444
09 kgdb_breakpoint (new_dbg_io_ops=<value optimized out>)
10 at /build/foo/linux-2.6.38/kernel/debug/debug_core.c:960
11 960      wmb(); /* Sync point after breakpoint */
12
13 (gdb)
```

Breakpoints

- Anlegen mit
 - b** [**<Dateiname>**:]**<Funktionsname>**
 - b** **<Dateiname>**:**<Zeilennummer>**
 - b** ***<Adresse>**Breakpoint im open-Systemaufruf: **b** `__x64_sys_open`
- Fortfahren der Ausführung mit **c** (continue)
- Schrittweise Abarbeitung auf Ebene der Quellsprache mit
 - s** (step) läuft in Funktionen hinein
 - n** (next) behandelt Funktionsaufrufe als einzelne Anweisung
 - finish** läuft bis zum Ende der aktuellen Funktion (nützlich wenn man versehentlich **s** statt **n** verwendet hat)
- Breakpoints anzeigen: **info breakpoints**
- Breakpoint löschen: **delete breakpoint**

p expr Anzeigen von Variablen (expr ist ein C-Ausdruck, im einfachsten Fall der Name einer Variable)

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

set <variablenname>=<wert> Setzen von Variablenwerten

bt (backtrace) Ausgabe des Funktionsaufruf-Stacks

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

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

Fragen?