Praktikum angewandte Systemsoftwaretechnik (PASST)

Arbeitsumgebung / Aufgabe 1

21. Oktober 2019

Tobias Langer, 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

Semester*ferien*

Blockpraktikum (5 ECTS)

Gruppenweises Bearbeiten einer umfangreichen Aufgabe (Projektcharakter)

Möglicher Termine:

- 17. Februar bis 28. Februar 2020 (KW 8 & 9)
- 24. Februar bis 06. März 2020 (KW 9 & 10)

Festlegung *nächste Woche*

Motivation

Motivation

- Linux ist quelloffen und kann „einfach“ erweitert werden
- Bei der Entwicklung passieren immer Fehler
- Neuer Code sollten ausführlich getestet & vermessen werden
- Aber …
  - … wie Systemzustand untersuchen, ohne ihn zu beeinflussen?
  - … wie Fehler suchen, wenn das OS hängt?
  - … Neustart und Einspielen eines neuen Systems dauert

Motivation

- Linux ist quelloffen und kann „einfach“ erweitert werden
- Bei der Entwicklung passieren immer Fehler
- Neuer Code sollten ausführlich getestet & vermessen werden
- Aber …
  - … wie Systemzustand untersuchen, ohne ihn zu beeinflussen?
  - … wie Fehler suchen, wenn das OS hängt?
  - … Neustart und Einspielen eines neuen Systems dauert

Es werden spezielle Testumgebungen und Werkzeuge benötigt!

Lernziele

Lernziele

Im Anschluss an diese Aufgabe solltet Ihr...

- das Konfigurationssystem des Linuxkerns beschreiben
- den Linuxkern konfigurieren und übersetzen
- eigenständig eine Umgebung für Arbeit und Tests am Linuxkern einrichten
- mit KGDB einen laufenden Linuxkern debuggen

...können.

Agenda

Arbeitsumgebung einrichten

Eigenen Linuxkern übersetzen & ausführen

Konfiguration & Arbeit mit dem Kernel Debugger

Zusammenfassung

Aufgabe 1

Arbeitsumgebung einrichten

Virtualisierung

- 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

Virtuelle Festplatte anlegen

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

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

Minimales Installationsprogram laden

Download des Debian netinst Installers

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

- Minimaler Debian Installer (nur wenige MiB groß)
  - Kernel + Ramdisk (Installationsprogramm).
- Alles Weitere wird vom Debian Spiegel nachgeladen.

Starten der virtuellen Maschine mit KVM

Starten der virtuellen Maschine

```
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 nur über serielle Konsole (`/dev/ttyS0`)
- *Empfehlung*: Wiederverwendbares (Bash-)Skript

```
$ cat boot.sh
kvm -m 1024 ... "$@"
$ ./boot.sh -kernel linux -append "console=ttyS0"
```

Qemu/KVM Kommandos

Kommandos von Qemu/KVM werden durch die Escape Sequence Ctrl + a eingeleitet:

- 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.

Installation im Textmodus

Starten der Installation in der virtuellen Maschine

```
$ ./boot.sh -kernel linux -initrd initrd.gz \
-append "console=ttyS0 priority=low"
[    0.000000] Linux version 4.19.0-6-amd64 (debian-kernel...
[    0.000000] Command line: console=ttyS0 priority=low
[    0.000000] x86/fpu: x87 FPU will use FXSAVE
[    0.000000] BIOS-provided physical RAM map:
[    0.000000] BIOS-e820: [mem 0x0000000000000000-0x000000...
...
```

FAU Debian Spiegel

Für die Installation den https Spiegel `ftp.fau.de` wählen!

Testen der Installation

- Testen ob die Installation fehlerfrei verlaufen ist
  - Nutzer können sich einloggen
  - Programme lassen sich wie gewohnt starten
  - Zugriff auf Netzwerk funktioniert
  - ...

Fallstricke

- Debian netinst Kernel unterstützt keine VirtIO Dateiträger
  - → Testkernel: `/proj/i4passt/kernel/linux_test`
- Netzwerk wird per NAT zur Verfügung gestellt
  - → Gast & Host haben nach außen die gleiche IP
- VM wird als Gast ausgeführt
  - → Befehle wie `ping` funktionieren nicht

Eigenen Linuxkern übersetzen & ausführen

Quellen vorbereiten

Klonen der Linuxquellen von den i4 Quellen

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

- Konfiguration von Linux mittels

  ```
  make menuconfig
  ```

  oder

  ```
  make xconfig
  ```

- Bauen mittels

  ```
  make -j $(nproc)
  ```

Kernel-Optionen

- Nützliche Optionen

  `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 deaktivieren:
  - Firewire, Sound, Multimedia, obskure Netzwerkprotokolle...

Arbeiten im CIP

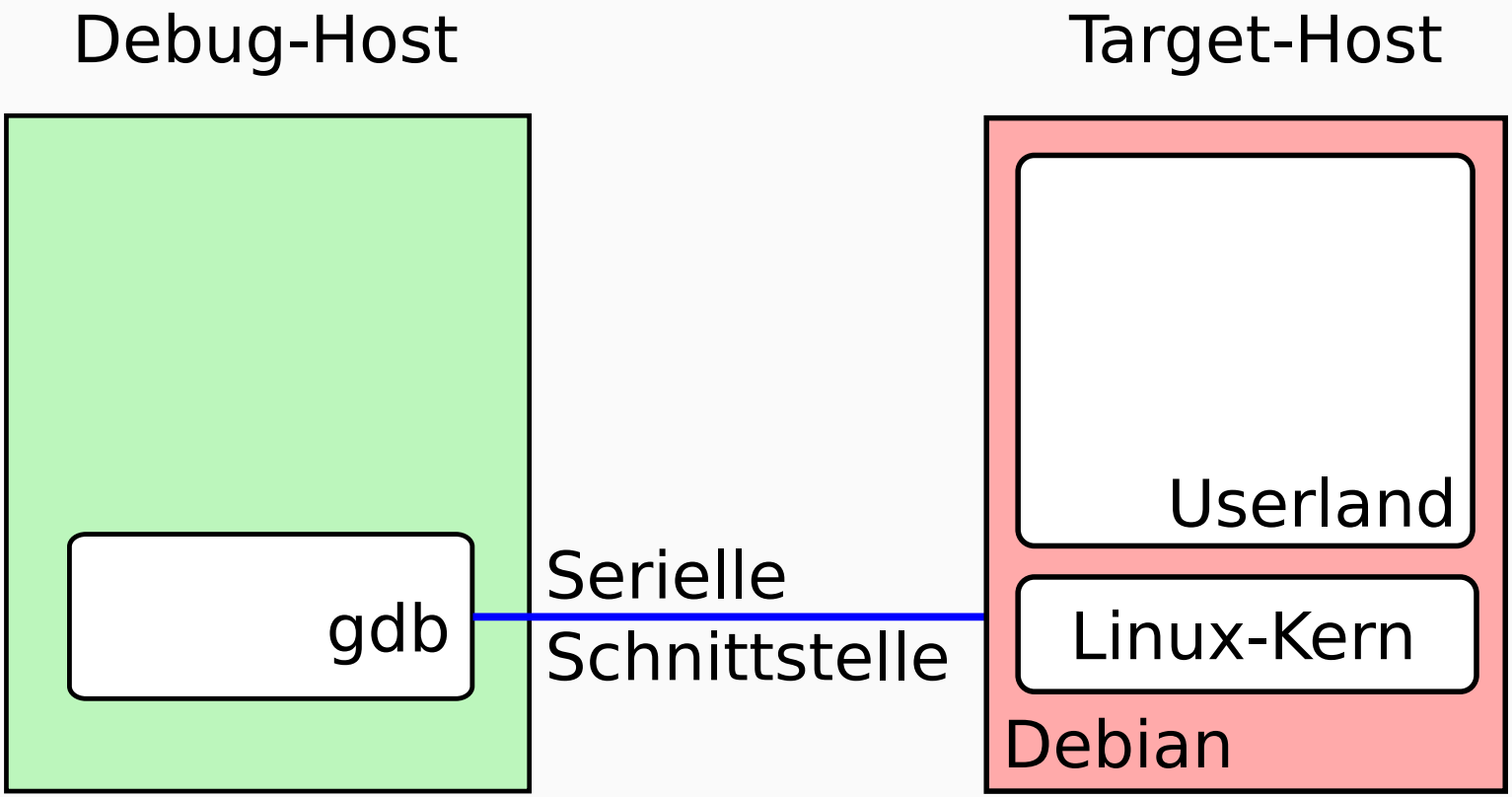
- Ablage & bauen des Linuxkerns unter `/var/tmp`
  - Ablage auf lokaler Festplatte
  - Daten sind nur lokal verfügbar
  - kein Backup
- Ablage & Installation der VM unter `/proj/i4passt`
  - Verzeichnisse unter `students/<Kennung>`
  - NFS Mount $\rightsquigarrow$ Nicht unbedingt zum Bauen geeignet
  - Nur begrenzt Platz

Wichtig: `/var/tmp` wird regelmäßig bereinigt!

$\rightarrow$ Sicherung der Änderungen notwendig (bspw. als Patchset)

Konfiguration & Arbeit mit dem Kernel Debugger

Logisches Debugsetup



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

GRUB Konfiguration (`/etc/default/grub`)

```
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"
```

Debuggen mit dem KGDB

- 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
```

Remote Debugging mit KGDB

```
$ 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 von Breakpoints mit `break`

  ```
  break [<Dateiname>:]<Funktionsname>
  break <Dateiname>:<Zeilennummer>
  break *<Adresse>
  ```

- Breakpoints anzeigen

  ```
  info breakpoints
  ```

- Breakpoint löschen

  ```
  delete breakpoint
  ```

- Beispiel: Breakpoint im `open`-Systemaufruf

  ```
  break do_sys_open
  ```

Schrittweise durch das Programm

- Fortführen der Ausführung nach Breakpoint

  `continue`
- Ausführen bis zur nächsten Zeile (Betritt Funktionen)

  `step`
- Ausführen bis zur nächsten Zeile (Betritt Funktionen *nicht*)

  `next`
- Ausführen bis zum Ende der aktuellen Funktion

  `finish`

Für einzelne Instruktionen analog mit `stepi`, `nexti`

Variablen, Stack

- Anzeigen von Variablen

  `print expr` (`expr` kann auch C-Ausdruck sein)
- Anzeigen von Variablen bei jedem Stopp (Breakpoint, Step, ...)

  `display expr`
- Setzen von Variablenwerten

  `set <variablenname>=<wert>`
- Ausgabe des Aufrufstacks

  `backtrace`

Watchpoints

Watchpoints stoppen Ausführung bei Zugriff auf eine Variable

- Stopp, wenn sich Wert des C-Ausdrucks `expr` ändert

  `watch expr`
- Stopp, wenn `expr` gelesen wird

  `rwatch expr`
- Stopp, bei jedem Zugriff (kombiniert `watch` und `rwatch`)

  `awatch expr`

Anzeigen und Löschen analog zu den Breakpoints

Weitere Tipps & Dokumentation

- (Fast) alle Kommandos lassen sich abkürzen

  `b` break

  `s` step

  `ni` nexti

- History in Konfiguration aktivieren

  ```
  set history save on
  ```

- GDB TUI: Verschiedene Ansichten im Terminal

  ```
  tui enable
  layout {src,asm,split,regs}
  ```

- Weitere Infos:
  - GDB Dokumentation
  - Kernelspezifisch: Documentation/dev-tools/gdb-kernel-debugging.rst

Zusammenfassung

Zusammenfassung

- Virtualisierung bspw. mit QEMU/KVM
  - → Leichteres Debugging
  - → Kürzere Umlaufzeiten
- Konfigurieren und bauen des Linuxkernels
  - → Konfiguration über `Kconfig`
- Debugging mit KGDB
  - → Remote Debugging
  - → Linux bietet spezielle Debugoptionen an

Aufgabe 1

Arbeitsumgebung aufsetzen und Einrichten (1)

1. Virtuelle Maschine einrichten & Installieren
2. Quellen des Linuxkern beziehen, konfigurieren & übersetzen
3. Virtuelle Maschine mit eigenem Kernel starten
4. Mit KGDB Breakpoints in virtueller Maschine mit eigenem Linux setzen & testen
5. ...gegebenenfalls Skripte für wiederkehrende Aufgaben erstellen.

Wichtig

Ihr solltet die Schritte erklären können (bspw. gesetzte Optionen in KVM/beim Bauen)

Arbeitsumgebung aufsetzen und Einrichten (2)

Wettbewerb: Wer schafft den kleinsten Kernel?

- Kernel muss fehlerfrei übersetzen
- Kernel muss bootbar & funktionstüchtig sein

Fragen?