Praktikum angewandte Systemsoftwaretechnik (PASST)

Arbeitsumgebung / Aufgabe 1

2. November 2020

Dustin Nguyen, Tobias Langer, Jonas Rabenstein, Phillip Raffeck

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

Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

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 System 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 System 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 Linux-Kernels beschreiben
- den Linux-Kernel konfigurieren und übersetzen
- eigenständig eine Umgebung für Arbeit und Tests am Linux-Kernel einrichten
- mit **KGDB** einen laufenden Linux-Kernel debuggen

...können.

Agenda

Arbeitsumgebung einrichten

Eigenen Linux-Kernel übersetzen & ausführen

Arbeiten mit der VM

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.

- Folien & Übung benutzen standardmäßig **QEMU**/**KVM**

Virtuelle Festplatte vorbereiten

Anlegen der virtuallen Festplatte:

```
$ 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**)

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 -nodefaults -nographic \
-m 1024 -serial mon:stdio \
-serial tcp::9876,server,nowait,nodelay \
-net nic,model=virtio -net user,hostfwd=tcp:127.0.0.1:5022-:22 \
-drive file=passt.img,if=virtio,cache=writeback,format=raw \
-kernel <kernel_binary> [args...]
```

Starten der virtuellen Maschine mit KVM

Starten der virtuellen Maschine:

```
$ kvm -nodefaults -nographic \
-m 1024 -serial mon:stdio \
-serial tcp::9876,server,nowait,nodelay \
-net nic,model=virtio -net user,hostfwd=tcp:127.0.0.1:5022-:22 \
-drive file=passt.img,if=virtio,cache=writeback,format=raw \
-kernel <kernel_binary> [args...]
```

- `-m`: RAM Größe

Starten der virtuellen Maschine mit KVM

Starten der virtuellen Maschine:

```
$ kvm -nodefaults -nographic \
-m 1024 -serial mon:stdio \
-serial tcp::9876,server,nowait,nodelay \
-net nic,model=virtio -net user,hostfwd=tcp:127.0.0.1:5022-:22 \
-drive file=passt.img,if=virtio,cache=writeback,format=raw \
-kernel <kernel_binary> [args...]
```

- `-m`: RAM Größe
- `-nodefaults`: Keine Standart-Geräte (CD-/Floppylaufwerk, ...)
- `-nographic`: Keine Graphik-Emulation

  $\rightarrow$ Start als Kommandozeilenanwendung

Starten der virtuellen Maschine mit KVM

Starten der virtuellen Maschine:

```
$ kvm -nodefaults -nographic \
-m 1024 -serial mon:stdio \
-serial tcp::9876,server,nowait,nodelay \
-net nic,model=virtio -net user,hostfwd=tcp:127.0.0.1:5022-:22 \
-drive file=passt.img,if=virtio,cache=writeback,format=raw \
-kernel <kernel_binary> [args...]
```

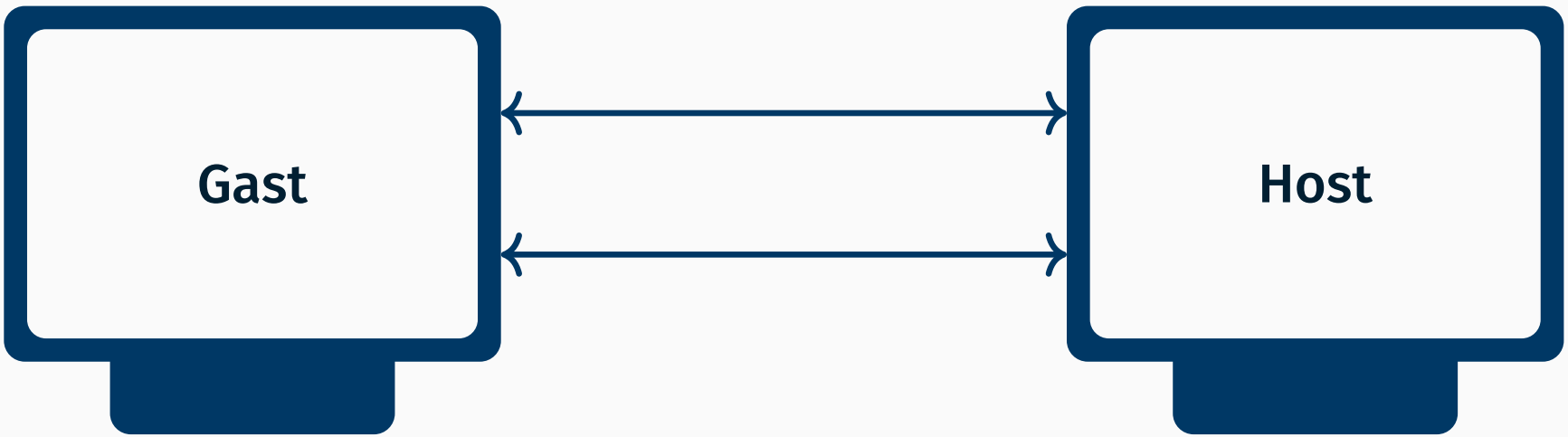
- `-m`: RAM Größe
- `-nodefaults`: Keine Standart-Geräte (CD-/Floppylaufwerk, ...)
- `-nographic`: Keine Graphik-Emulation

  $\rightarrow$ Start als Kommandozeilenanwendung
- `-net`: Paravirtualisiertes Netzwerk-Gerät
- `-drive`: Paravirtualisiertes Block-Gerät

Starten der virtuellen Maschine mit KVM

Starten der virtuellen Maschine:

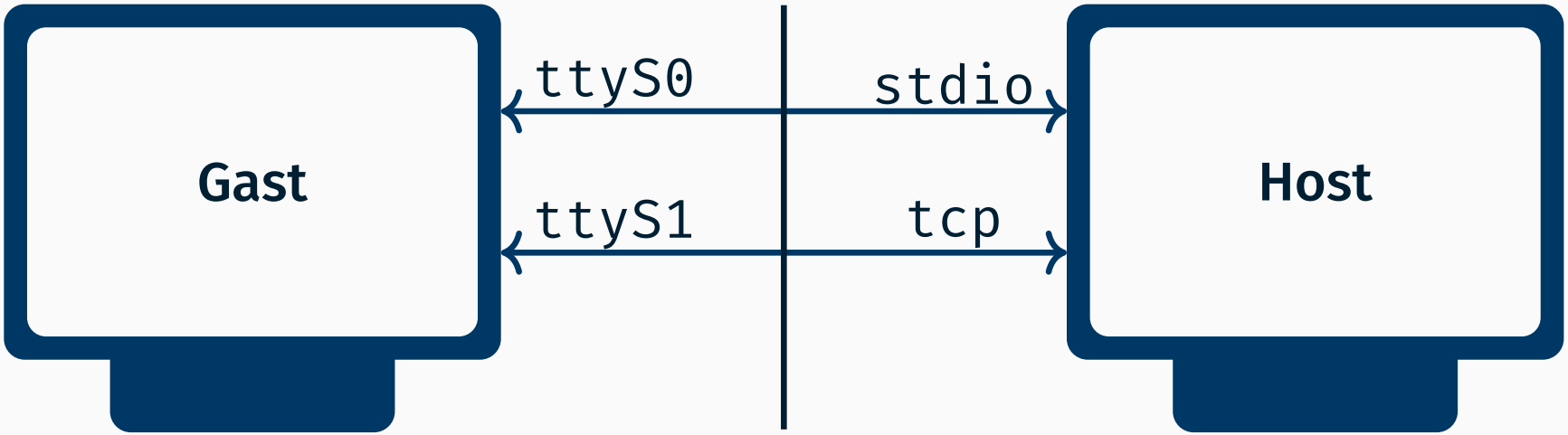
```
$ kvm -nodefaults -nographic \
-m 1024 -serial mon:stdio \
-serial tcp::9876,server,nowait,nodelay \
-net nic,model=virtio -net user,hostfwd=tcp:127.0.0.1:5022-:22 \
-drive file=passt.img,if=virtio,cache=writeback,format=raw \
-kernel <kernel_binary> [args...]
```



Starten der virtuellen Maschine mit KVM

Starten der virtuellen Maschine:

```
$ kvm -nodefaults -nographic \
-m 1024 -serial mon:stdio \
-serial tcp::9876,server,nowait,nodelay \
-net nic,model=virtio -net user,hostfwd=tcp:127.0.0.1:5022-:22 \
-drive file=passt.img,if=virtio,cache=writeback,format=raw \
-kernel <kernel_binary> [args...]
```



Start-Skript

- *Empfehlung*: Wiederverwendbares (Bash-)Skript

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

- Über Kommandozeile werden die Bootparameter übergeben:

  `-kernel` Pfad zu Kernel

  `-append` Kernelparameter

  `-initrd` Bei Bedarf: Pfad zur Initramfs

- Nützliche Kernelparameter

  `console=ttyS0` Konsole des Kernel

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

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 5.6.0-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 0x0000000000000000-0x000...
...
```

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

Eigenen Linux-Kernel 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 # ncurses
  ```

  oder

  ```
  $ make xconfig    # GTK+
  ```

- Bauen mittels

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

  oder

  ```
  $ make CC="ccache gcc" -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 Linux-Kernels 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 $\rightarrow$ Nicht unbedingt zum Bauen geeignet
  - Nur begrenzt Platz

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

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

Arbeiten mit der VM

SSH-Authentifizierung mit einem Schlüsselpaar ohne Passwort

- `ssh-keygen` erzeugt **privaten** und **öffentlichen** Schlüssel

```
$ ssh-keygen -f <name> -N ""
Generating public/private rsa key pair.
Your identification has been saved in <name>.
Your public key has been saved in <name>.pub.
[...]
```

- Generierte Schlüssel (`name = gruppe0`)

```
$ ls -l
-rw------- 1 langer users  1675  8. Mai 11:29 gruppe0
-rw-r--r-- 1 langer users   394  8. Mai 11:29 gruppe0.pub
```

In der Betriebsumgebung des Host-Rechners ausführen!

SSH-Authentifizierung mit Schlüsseln (VM)

- Installation des SSH-Servers in der virtuellen Maschine

```
$ apt-get install ssh openssh-server
```

- Zugriffe auf die virtuelle Maschine unter Zuhilfenahme des generierten **öffentlichen** Schlüssels
- Hinterlegen des **öffentlichen** Schlüssels

```
$ su - <vm_user>
$ mkdir .ssh
$ scp <user>@<host_ip>:~/<gruppen_name>.pub \
      /home/<vm_user>/.ssh/authorized_keys
```

In der Betriebsumgebung der virtuellen Machine ausführen!

Alternative: `ssh-copy-id(1)`

Verbindungsaufbau vom Host zur virtuellen Maschine

- Kontrollverbindung zur virtuellen Maschine aufbauen

```
$ ssh -p 5022 -i <name> <vm_user>@localhost
```

- Datenverbindung

```
$ scp -P 5022 -i <name> <datei1> [<datei2>] \
<vm_user>@localhost:/<vm_path>
```

- Alternative: SSHFS (benötigt root-Rechte!)

```
$ sshfs -p 5022                                  \
-o IdentityFile=<absolute_path>/<gruppe_name> \
<vm_user>@localhost:/<vm_path>                   \
<mount_point>
```

Hinweis bei Verwendung von `sshfs(1)`: Absoluter Pfad zum Schlüssel zwingend notwendig

Konfiguration & Arbeit mit dem Kernel Debugger

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*.

Debuggen mit dem KGDB

- Unterbrechen funktioniert nicht via **KGDB**.
- Stattdessen aus der VM:

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

- Nützliche Kernelparameter

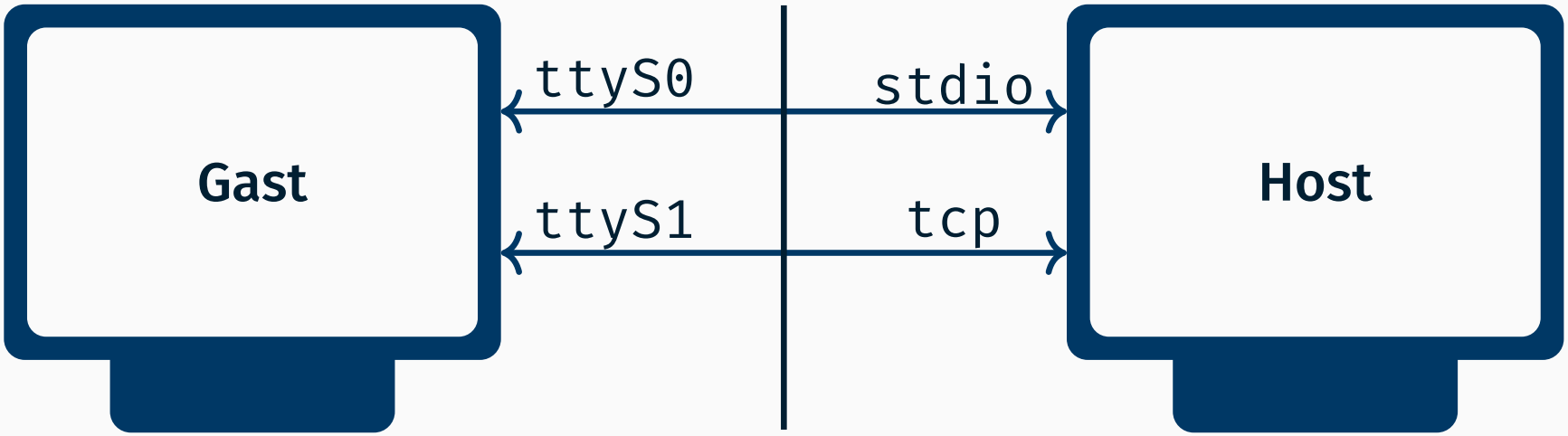
`kgdboc=ttyS1,115200`	KGDB konfigurieren
`kgdbwait`	Beim Booten auf eine GDB-Verbindung warten

Debuggen mit dem KGDB

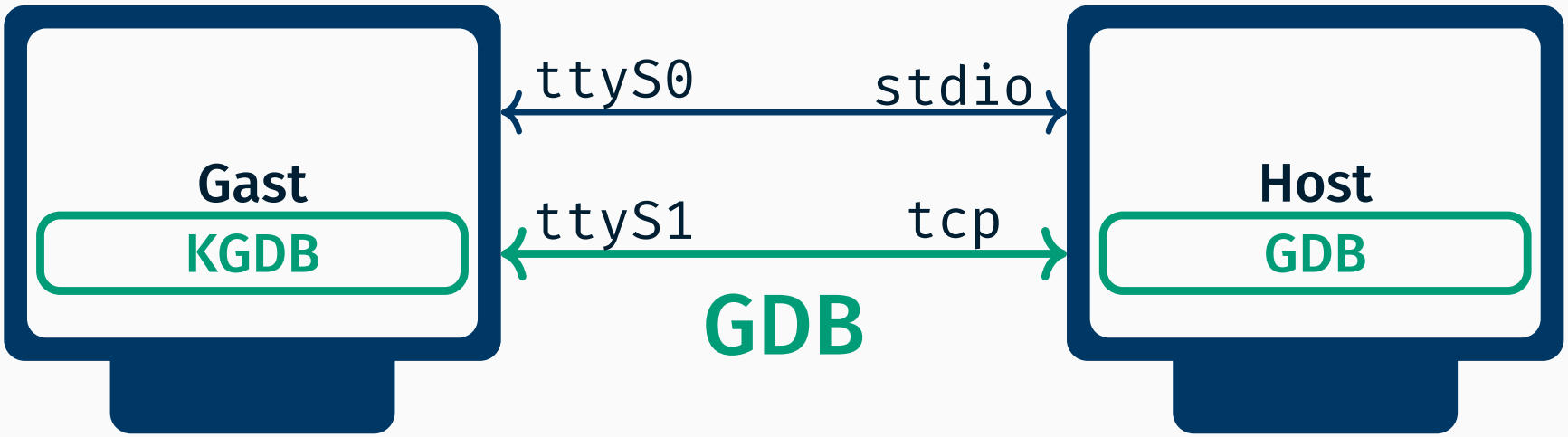
Remote Debugging mit KGDB:

```
$ gdb vmlinux
[...]
Reading symbols from /build/foo/linux-5.6.0/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-5.6.0/kernel/debug/debug_core.c:960
960         wmb(); /* Sync point after breakpoint */

(gdb)
```

Logisches Debugsetup



Logisches Debugsetup



Breakpoints

- Anlegen von Breakpoints mit `break`

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

- Anlegen von Hardware-Breakpoints mit `hbreak`

```
(gdb) hbreak ... # analog zu break
```

- Breakpoints anzeigen

```
(gdb) info breakpoints
```

- Breakpoint löschen

```
(gdb) delete breakpoint
```

- Beispiel: Breakpoint im open-Systemaufruf

```
(gdb) break do_sys_open
```

Schrittweise durch das Programm

- Fortführen der Ausführung nach Breakpoint
  `(gdb) continue`
- Ausführen bis zur nächsten Zeile (Betritt Funktionen)
  `(gdb) step`
- Ausführen bis zur nächsten Zeile (Betritt Funktionen *nicht*)
  `(gdb) next`
- Ausführen bis zum Ende der aktuellen Funktion
  `(gdb) finish`

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

Variablen, Stack

- Anzeigen von Variablen bzw. C-Ausdrücke

  ```
  (gdb) print expr
  ```

- Anzeigen von Variablen bei jedem Stopp (Breakpoint, Step, ...)

  ```
  (gdb) display expr
  ```

- Setzen von Variablenwerten

  ```
  (gdb) set <variablenname>=<wert>
  ```

- Ausgabe des Aufrufstacks

  ```
  (gdb) backtrace
  ```

Watchpoints

Watchpoints stoppen Ausführung bei Zugriff auf eine Variable

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

Anzeigen und Löschen analog zu den Breakpoints

Weitere Tipps & Dokumentation

- (Fast) alle Kommandos lassen sich abkürzen

```
(gdb) b  # break
(gdb) hb # hardware break
(gdb) s  # step
(gdb) ni # nexti
```

- History in Konfiguration aktivieren

```
(gdb) set history save on
```

- GDB TUI: Verschiedene Ansichten im Terminal

```
(gdb) tui enable
(gdb) 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 Linux-Kernels
  - → Konfiguration über **`Kconfig`**
- Debugging mit **KGDB**
  - → Remote Debugging
  - → Linux bietet spezielle Debugoptionen an

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...

Aufgabe 1

Arbeitsumgebung aufsetzen und Einrichten (1)

1. Virtuelle Maschine einrichten & Installieren
2. Quellen des Linux-Kernels 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?