Linux Kernel Debugging

Seminar "Effiziente Programmierung"

Leon Fritz

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

28.01.2021

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

informatik die zukunft

Gliederung

1 Motivation

2 Überblick

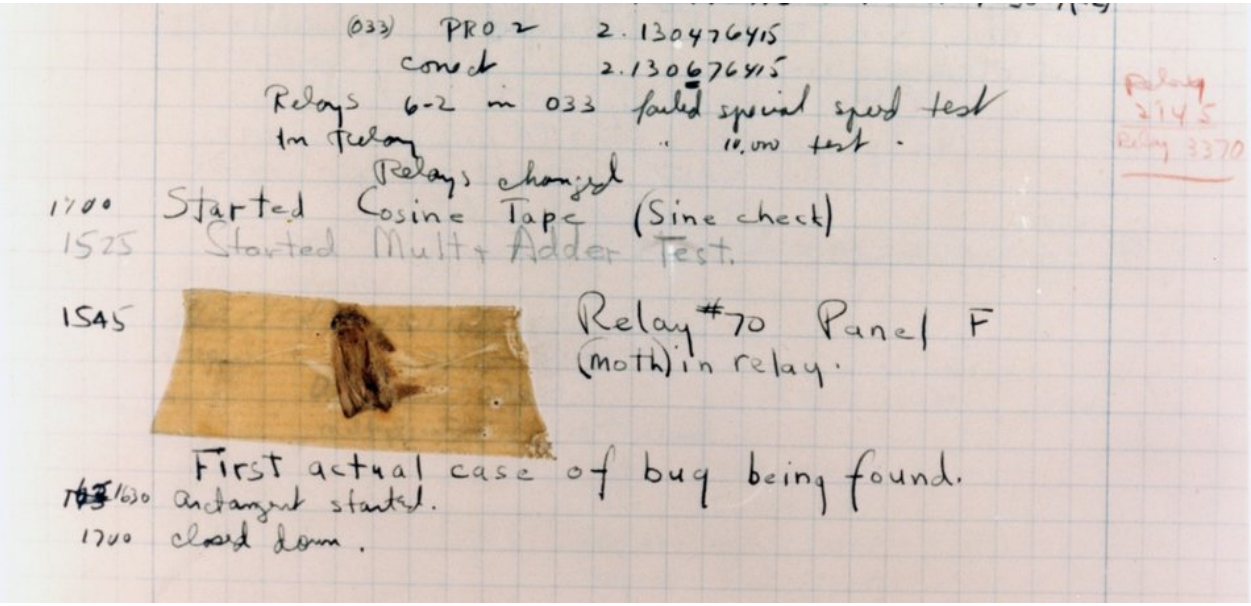
3 Debugging Tools

4 Zusammenfassung

5 Literaturverzeichnis

Einleitung

- Wofür steht Linux Kernel Debugging?
- Warum verwenden wir Linux Kernel Debugging Tools?



[1]

Warum ein Linux Kernel Debug Tool nutzen?

- Hilfreich beim Entwickeln des Kernels oder Kernel Modules
- Fehler im Kernel erkennen
- Fehler in Kernel Modules (z.B. Treibern) erkennen
- Sicherheitslücken und Bugs finden, die durch den Compiler entstanden sind
- Besseres Verständnis über das Zusammenwirken des Linux-Kernels bilden

Logik (unser Kopf)

- ➕Vor allem wenn, man sich mit der Anwendung auskennt, kann es zu schnellen Lösungen führen
- ➕Je nach Kenntnisgrad können auch kompliziertere Probleme gefunden werden
- ⛔Ist nur so gut wie unser Wissen über den Code

[2]

Logbasiertes Debugging

- ➕Extrem verständlich selbst für Laien
- ➕Gibt konkrete Fehlermeldungen aus
- ⛔Setzt voraus, dass der Entwickler Vorarbeit geleistet haben muss
- ⛔Erfordert je nach Art (z.B. Minidumps) spezielle Parserprogramme
- ⛔Möglich erst nachdem ein Problem aufgetreten ist

Quellcode-basiertes Debugging

- ➕Hohe Benutzerfreundlichkeit, debuggbar für Entwickler der die Kernelprogrammiersprache
- ⛔Setzt Verfügbarkeit des Quellcodes voraus
- ⛔Abstrahiert den gesamten Kompilierungsvorgang
  - Führt zu versteckten Fehlern

Quellcode-loses Debugging

- ➕Der Abstraktionsgrad ist gering
  - Keine versteckten Fehler
- ➕Auch Programme und Kernelblobs ohne Quellcode debugbar
- ⛔Schwer zu verstehen, aufgrund des geringen Abstraktionsgrades
- ⛔Steile Lernkurve
- ⛔Viel Hintergrundwissen notwendig
  - x86 Assemblersprache
  - x64 Assemblersprache
  - C (teils C++)

Arten von Debuggern

- Automatische Frameworks (z.B. AFL Fuzzer)
  - ➕Wenig Arbeit des Entwicklers gefordert
  - ➕Sehr hilfreich zum finden neuartiger Crashes bevor sie Schäden anrichten
  - ⛔Verglichenerweise ein sehr zeitintensives Vorgehen
  - ⛔Kann trotz genügend Zeit nicht alle Code-Paths finden
  - ⛔Entwickler muss das Framework genau auf die Codestelle ansetzen
  - ⛔Entwickler muss trotzdem die Ergebnisse interpretieren

Arten von Debuggern

- Profiling/Tracing
  - ➕Leistungseinbußen sind leicht erkennbar
  - ⛔Je nach Tool viel Arbeit vom Entwickler gefordert
  - ⛔Hilft nur bei Leistungseinbußen und Lockups

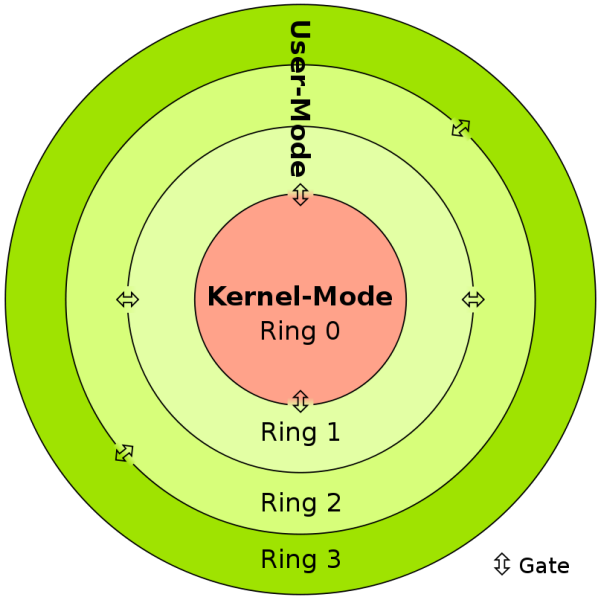
Debugging Schrittweise

1. Sich bewusst machen, dass es ein Problem gibt
2. Dieses Problem reproduzierbar machen
3. Das Problem genauer betrachten
4. Die Ursache des Problems finden
5. Das Problem lösen, indem man die Ursache des Problems bekämpft!

[3]

Kernel im Vergleich zum Userspace

- Zugriffsbeschränkungen
- Ringe/Priviledges [4]
- Syscalls
- Konsequenzen für Fehler [5]
- Portabilität
- Entwicklerzeit
- Debugging



[6]

Probleme im Kernel

- Crashes [7]
- Freezes
- Logikfehler
- Resource Leaks (z.B. Memory)
- Performance Issues

Fatalität der Probleme

- Soft-Lockups [8]
- Hard-Lockups
- Kernel-Oops [9]
- Kernel-Panic [20]

[10]

Vergleich von Problemen und Debug-Techniken

*	Crashes	Locks	Logik	Leaks	Bottlenecks
Wissen	+	+	+	+	+
Logbasiertes Debugging	+	−	−	⊖	⊖
Quellcode-basiertes Debugging	+	+	+	−	⊖
Quellcode-loses Debugging	+	+	+	−	⊖
Profiling/Tracing	−	+	−	−	+
Automatische Frameworks	−	−	⊖	−	−

SysRq Key

- Funktioniert auf jedem Linux-System
- Funktioniert in-situ
- Vielzahl von Kommandos
  - Generieren von Kernel Dumps
  - Eliminieren von Vorgängen
  - Anzeigen von diversen Debug-Information
  - Neu-Initialisierung und Synchronisation von Dateisystemen
  - Öffnen von KDB

  [11]
- Erlaubt oftmals Rettung von Daten [12]

[13]

SysRq Key

Live-Demo

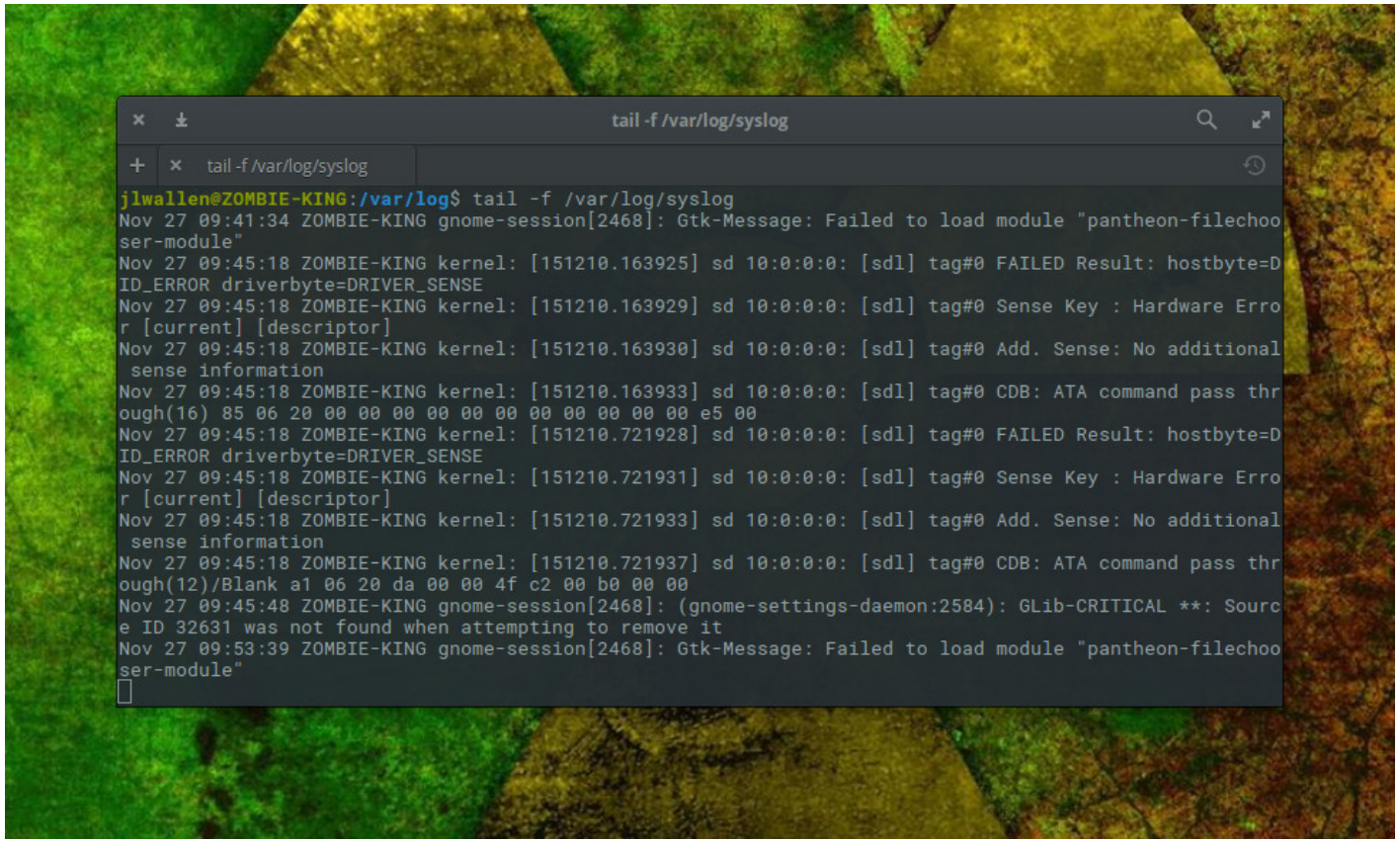
Kernel Logs

- Nutzung nur post mortem möglich
- Funktioniert auf jedem Linux-System
- Ermöglicht es, rückblickend Fehler zu analysieren
- Ermöglicht Einblick in Kernel-Funktionalität
- Text mit printk hinzufügbar
- Auch für Laien möglich

[14]

Kernel Logs

Live-Demo



Kernel Dumps

- Nutzung nur post mortem möglich (trotz dessen teils in-situ)
- Nicht bei jeder Distribution standardmäßig aktiviert
- Ermöglicht es, rückblickend Fehler zu analysieren
- Enthält eine Vielzahl an Debug-Information
- Ist sicher, da ein Zweitkernel den Dump generiert

[15]

Kernel Dumps

admin@testhost2:/var/crash/2010-01-19-20:21

File Edit View Terminal Tabs Help

```
    exception stack pointer: ffff810107132f20
      process stack pointer: ffff81010712bef0
         current_stack_base: ffff8101b509c000

      KERNEL: /usr/lib/debug/lib/modules/2.6.18-164.10.1.el5.centos.plus/vmlinux
    DUMPFILE: vmcore
        CPUS: 2
        DATE: Tue Jan 19 20:21:19 2010
      UPTIME: 00:00:00
LOAD AVERAGE: 0.00, 0.04, 0.07
       TASKS: 134
    NODENAME: testhost2
     RELEASE: 2.6.18-164.10.1.el5
     VERSION: #1 SMP Thu Jan 7 19:54:26 EST 2010
     MACHINE: x86_64  (3000 Mhz)
      MEMORY: 7.5 GB
       PANIC: "SysRq : Trigger a crashdump"
         PID: 0
     COMMAND: "swapper"
        TASK: ffffffff80300ae0  (1 of 2)  [THREAD_INFO: ffffffff803f2000]
         CPU: 0
       STATE: TASK_RUNNING (ACTIVE)

crash>
```

Ftrace

- Sehr komfortabel, da von Userspace aus nutzbar
- Funktioniert in-situ
- Einblick in eine Vielzahl von Trace-Punkten

[16]

Ftrace

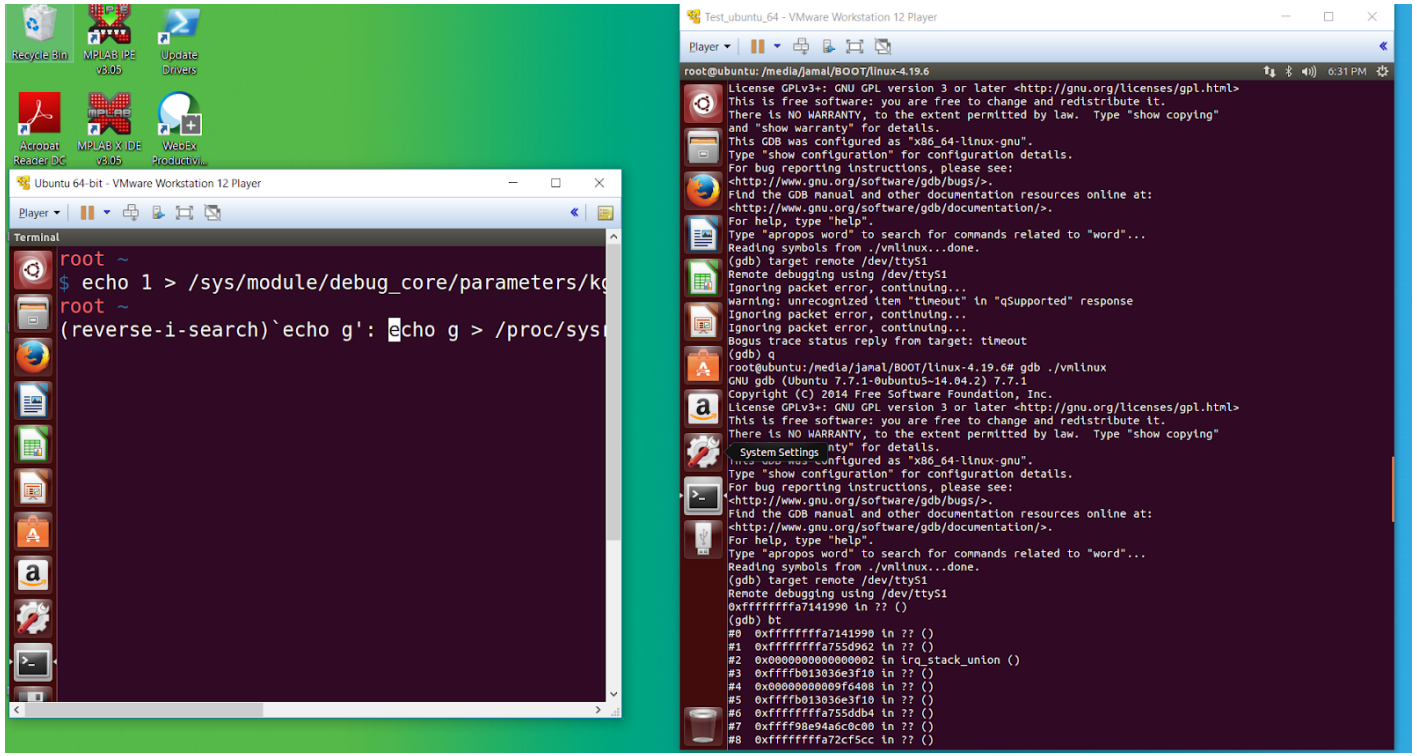
```
root ftrace
$ ./kernel_functions.sh
Battery 0: Discharging, 65%, 04:12:19 remaining
Battery 0: design capacity 7840 mAh, last full capacity 7648 mAh = 97%
Adapter 0: off-line
Cooling 0: Processor 0 of 7
Cooling 1: Processor 0 of 7
Cooling 2: Processor 0 of 7
Cooling 3: Processor 0 of 7
root ftrace
$ cat /sys/kernel/tracing/trace | head -n 15
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
  3)   1.076 us    |  mutex_unlock();
  3)   0.155 us    |  __fsnotify_parent();
  3)   0.120 us    |  fsnotify();
  3)   0.103 us    |  __sb_end_write();
  3)               |  do_syscall_64() {
  3)               |    __x64_sys_dup2() {
  3)               |      ksys_dup3() {
  3)   0.125 us    |        _raw_spin_lock();
  3)   0.116 us    |        expand_files();
  3)               |        do_dup2() {
  3)               |          filp_close() {
root ftrace
```

(K)GDB

- Extrem detailliert
- Schwer für Laien nutzbar
- Benötigt 2 Maschinen
  - Host-Maschine startet GDB mit Symbolen
  - Ziel-Maschine enthält den Kernel der analysiert werden soll
- Funktioniert nur mit Vorkonfiguration in-situ
- Funktioniert für Quellcode-basiertes und Quelltext-loses Debugging
- Breakpoints nutzbar

[17]

(K)GDB


Recycle Bin
MPLAB IPE v3.05
Update Drivers
Acrobat Reader DC
MPLAB X IDE v3.05
WebEx Productivi...
Ubuntu 64-bit - VMware Workstation 12 Player
Player
Terminal
root ~
$ echo 1 > /sys/module/debug_core/parameters/kg
root ~
(reverse-i-search)`echo g': echo g > /proc/sys
Test_ubuntu_64 - VMware Workstation 12 Player
Player
root@ubuntu: /media/jamal/BOOT/linux-4.19.6
6:31 PM
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./vmlinux...done.
(gdb) target remote /dev/ttyS1
Remote debugging using /dev/ttyS1
Ignoring packet error, continuing...
warning: unrecognized item "timeout" in "qSupported" response
Ignoring packet error, continuing...
Ignoring packet error, continuing...
Bogus trace status reply from target: timeout
(gdb) q
root@ubuntu:/media/jamal/BOOT/linux-4.19.6# gdb ./vmlinux
GNU gdb (Ubuntu 7.7.1-0ubuntu5~14.04.2) 7.7.1
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
System Settings
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./vmlinux...done.
(gdb) target remote /dev/ttyS1
Remote debugging using /dev/ttyS1
0xffffffffa7141990 in ?? ()
(gdb) bt
#0 0xffffffffa7141990 in ?? ()
#1 0xffffffffa755d962 in ?? ()
#2 0x0000000000000002 in irq_stack_union ()
#3 0xffffb013036e3f10 in ?? ()
#4 0x00000000009f6408 in ?? ()
#5 0xffffb013036e3f10 in ?? ()
#6 0xffffffffa755ddb4 in ?? ()
#7 0xffff98e94a6c0c00 in ?? ()
#8 0xffffffffa72cf5cc in ?? ()

KDB

- Extrem detailliert
- Schwer für laien zu nutzen
- Funktioniert in-situ
- Benötigt nur die Maschine, die man gerade nutzt
- Funktioniert nur für Quelltext-loses Debugging
- Breakpoints nutzbar

[18]

KDB im Vergleich zu (K)GDB

- KGDB erlaubt Nutzung vom originalen Quellcode
- KGDB braucht 2 Maschinen während KDB nur eine braucht [19]
- Beide funktionieren in-situ [18]

Zusammenfassung

- Es gibt keine perfekte Möglichkeit fürs Kernel Debuggen
  - Jede Möglichkeit hat ihren Anwendungszweck
  - Nutze die richtigen Tools für den richtigen Zweck
- Probleme im Kernel sind fataler als im Userspace
- Probleme im Kernel sind schwerer zu debuggen

Literaturverzeichnis I

[1] https://www.heise.de/newsticker/meldung/Zahlen-bitte-Relais-70-defekt-Von-der-Geschichte-des-Bugs-4517093.html. Zugriff am: 26.01.2021.

[2] https://www.flaticon.com/. Zugriff am: 26.01.2021.

[3] https://www.flaticon.com/. Zugriff am: 26.01.2021.

[4] https://www.multicians.org/mgr.html#ring. Zugriff am: 26.01.2021.

[5] https://web.archive.org/web/20011215052223/http://uuu.sourceforge.net/si.php#SEC6. Zugriff am: 26.01.2021.

[6] https://de.wikipedia.org/wiki/Ring_(CPU). Zugriff am: 26.01.2021.

[7] https://www.unix.com/man-page/FreeBSD/9/panic/. Zugriff am: 26.01.2021.

Literaturverzeichnis II

[8] https://lists.kernelnewbies.org/pipermail/kernelnewbies/2012-January/004480.html. Zugriff am: 26.01.2021.

[9] http://madwifi-project.org/wiki/DevDocs/KernelOops. Zugriff am: 26.01.2021.

[10] https://www.pngkit.com/view/u2q8q8w7y3a9t4o0_sad-cute-penguin-sad-linux/. Zugriff am: 26.01.2021.

[11] https://www.kernel.org/doc/html/latest/admin-guide/sysrq.html. Zugriff am: 26.01.2021.

[12] https://www.howtogeek.com/119127/use-the-magic-sysrq-key-on-linux-to-fix-frozen-x-servers-cleanly-r
Zugriff am: 26.01.2021.

Literaturverzeichnis III

[13] https://www.howtogeek.com/119127/use-the-magic-sysrq-key-on-linux-to-fix-frozen-x-servers-cleanly-r
Zugriff am: 26.01.2021.

[14] https://linuxjourney.com/lesson/kernel-logging. Zugriff am: 26.01.2021.

[15] https://www.admin-magazin.de/Das-Heft/2012/04/Kernel-Crashdumps-aufsetzen-und-auswerten. Zugriff am: 26.01.2021.

[16] https://www.kernel.org/doc/Documentation/trace/ftrace.txt. Zugriff am: 26.01.2021.

[17] https://www.kernel.org/doc/html/v4.18/dev-tools/kgdb.html. Zugriff am: 26.01.2021.

Literaturverzeichnis IV

[18] https://bobcares.com/blog/the-kernel-debugging-tools-for-linux/. Zugriff am: 26.01.2021.

[19] http://oliveryang.net/2015/08/using-kgdb-debug-linux-kernel-1/#12-what-is-the-kdb-what-are-the-differences-between-kdb-and-kgdb. Zugriff am: 26.01.2021.

[20] M. Moore and S. Hancock. *Tru64 UNIX troubleshooting: diagnosing and correcting system problems*. Elsevier, 2002.