# Kernel-Debugging

Bernhard Walle
(bernhard@bwalle.de)

Chemnitzer Linux-Tage 2010

# Inhalt

# Kerneldokumentation



LXR linux/Documentation/kernel-parameters.txt - Mozilla Firefox

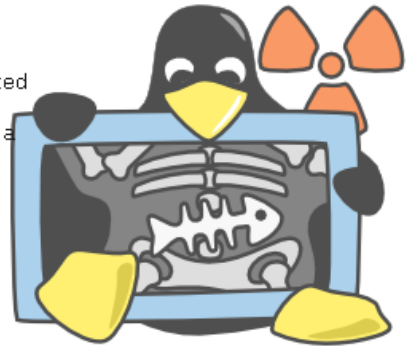
Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe

http://lxr.linux.no/#linux/Documentation/kernel-pa

LXR | linux/Documentation/kernel-parameters.txt

v2.6.32 Search Prefs

```
1                          Kernel Parameters
2                          ~~~~~~~~~~~~~~~~~~
3
4  The following is a consolidated list of the kernel parameters as implemented
5  (mostly) by the __setup() macro and sorted into English Dictionary order
6  (defined as ignoring all punctuation and sorting digits before letters in a
7  case insensitive manner), and with descriptions where known.
8
9  Module parameters for loadable modules are specified only as the
10 parameter name with optional '=' and value as appropriate, such as:
11
12         modprobe usbcore blinkenlights=1
13
14 Module parameters for modules that are built into the kernel image
15 are specified on the kernel command line with the module name plus
16 '.' plus parameter name, with '=' and value if appropriate, such as:
17
18         usbcore.blinkenlights=1
19
20 Hyphens (dashes) and underscores are equivalent in parameter names, so
21         log_buf_len=1M print-fatal-signals=1
22 can also be entered as
23         log-buf-len=1M print_fatal_signals=1
24
25
26 This document may not be entirely up to date and comprehensive. The command
27 "modinfo -p ${modulename}" shows a current list of all parameters of a loadable
28 module. Loadable modules, after being loaded into the running kernel, also
29 reveal their parameters in /sys/module/${modulename}/parameters/. Some of these
30 parameters may be changed at runtime by the command
31 "echo -n ${value} > /sys/module/${modulename}/parameters/${parm}".
32
33 The param
34 enabled a
35 the begir
36 parameter is applicable:
```



http://lxr.linux.no/#linux/

Fertig

# SysRq





`AltGr+Drucken+<Taste>`


© http://de.wikipedia.org/wiki/Tastatur

- Kernel: `CONFIG_MAGIC_SYSRQ=y`
- Einschalten: `sysctl kernel.sysrq=1`

ℹ `Documentation/sysrq.txt`

# Kerneloops

```
 EIP:    0060:[<c03ca9af>]    Not tainted VLI
 EFLAGS: 00010246   (2.6.8-prep)
 EIP is at find_isa_irq_pin+0x0/0x5d
 eax: 00000000   ebx: 00000000   ecx: 0000003f   edx: 00000003
 esi: c751f000   edi: 01234567   ebp: c751f000   esp: c751fe8c
 ds: 007b   es: 007b   ss: 0068
 Process reboot (pid: 3505, threadinfo=c751f000 task=d88b01b0)
 Stack: c011acf0 01234567 01234567 00000000 c751f000 01234567 c01172a3 00000000
        c0133d63 c0325a29 df6464b8 c13ee080 00c59fe0 d40d1ee8 00000001 dbf19480
        00c59fe0 dcd2300c d40d1ee8 c015700d 00000000 d848a164 dcd2300c dbf19480
 Call Trace:
  [<c011acf0>] disable_IO_APIC+0x16/0x1b6
  [<c01172a3>] machine_restart+0x6/0x6c
  [<c0133d63>] sys_reboot+0x19a/0x50f
  [<c015700d>] handle_mm_fault+0xe5/0x229
  [<c011ce75>] do_page_fault+0x1a5/0x4f4
  [<c01864b7>] destroy_inode+0x36/0x45
  [<c0181fab>] dput+0x33/0x4f3
  [<c0168a36>] __fput+0xc9/0xee
  [<c0167163>] filp_close+0x59/0x5f
  [<c0310c7b>] syscall_call+0x7/0xb
 Code: a2 f6 9f 5f e4 89 37 c8 78 47 c8 78 47 c8 78 47 9b 53 2b 9b 53 2b 8f 34 11
  6c 38 24 6d 2c 0c 4b 26 14 29 14 0f 67 4e 35 6d 2c 0c <63> 17 01 6d 2c 0c 46 13
  00 46 13 00 46 13 00 63 17 01 6d 2c 0c
  _
```

# Inhalt

# printk()





- Syntax:

```
int printk(const char *fmt, ...);
```

- Formatstring enthält »Loglevel«:

| Konstante | Wert | Bedeutung |
|---|---|---|
| KERN_EMERG | "<0>" | Unbenutzbares System |
| KERN_ALERT | "<1>" | Benutzerintervention unmittelbar erforderlich |
| KERN_CRIT | "<2>" | Kritischer Zustand |
| KERN_ERR | "<3>" | Fehler |
| KERN_WARNING | "<4>" | Warnung |
| KERN_NOTICE | "<5>" | Wichtige Meldung, aber kein Fehler |
| KERN_INFO | "<6>" | Informationsmeldung |
| KERN_DEBUG | "<7>" | Debugmeldung |

- Beispiel:

```
printk(KERN_INFO "%d devices found\n", nr_devices);
```

# printk() (cont.)

- Abkürzungsfunktionen, z. B.

```
pr_info("%d devices found", nr_devices);
```

- Ausgabe der Device-Struktur bei Treibern

```
int dev_printk(const char          *level,
               const struct device *dev,
               const char          *format, ...);
```

- Abkürzungsfunktionen, z. B.[1]

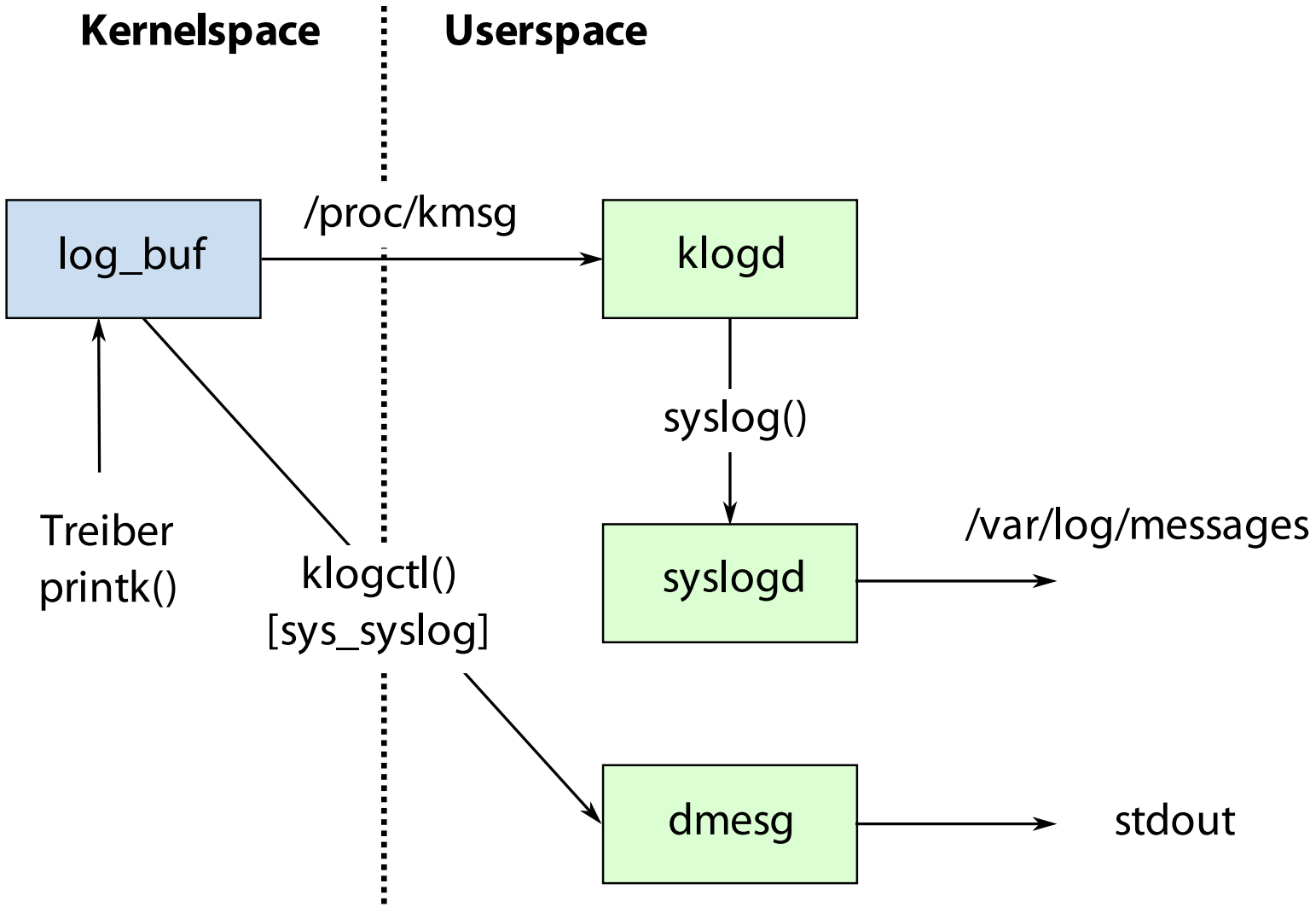
```
dev_info(&pdev->dev, "Prescaler is set at: %d\n",
         prsc);
```

- Zusicherungen (»Assertions«)

```
BUG_ON(count != 0);    /* loest panic() aus */
WARN_ON(count != 0);   /* gibt eine Warnung aus */
```

[1] aus drivers/input/touchscreen/atmel_tsadcc.c

# Kernellog-Mechanismus im System



# Kernel-Konsole





## Motivation

- `syslogd` und `dmesg` helfen nicht bei Abstürzen!

## Kernel-Konsole

- Standardmäßig die gerade aktive Konsole
- Manche Distributionen geben Kernelmeldungen auf extra Konsole aus (z. B. SUSE auf Konsole 10)
- Änderung mit `setlogcons` bzw. `klogconsole` möglich

## Loglevel für Kernelkonsole setzen

- `sysctl kernel.printk`
  (erste Zahl gibt den Loglevel auf der Konsole an)
- `SysRq-`*`Zahl`*

# Serielle Konsole





## Verbindung

- Nullmodemkabel

## Auf dem Entwicklungsrechner

- Serielle Schnittstelle erforderlich, kann auch USB-Adapter sein
- Terminalprogramm nötig, z. B. `minicom`, `screen` oder

```
% picocom -b 115200 /dev/ttyUSB0
```

## Auf dem Zielrechner

- Bei der Kernelkommandozeile hinzugfügen:

```
console=ttyS0,115200 console=tty0
```

- letzte Konsole wird zur »Systemkonsole«

# Serielle Konsole (cont.)





## Rechner bootet nicht

- Frühe Ausgaben können mit `earlyprintk` aktiviert werden, also beispielsweise `earlyprintk=ttyS0`

## Rechner hat keine serielle Schnittstelle

- PCI-Karte (10 €)
- Virtuelle serielle Konsole – *»Serial over LAN«*
  - IPMI (Intelligent Platform Management Interface)
  - iLO (HP Integrated Lights-Out)
  - AMT (Intel Active Management Technology)
- Netconsole

  ℹ `Documentation/networking/netconsole.txt`
- USB-Adapter (geht nicht immer)

# Inhalt

# Motivation

## Probleme beim `printk()`-Ansatz

- Ursache muss »ungefähr« bekannt sein
- Häufiges Neukompilieren des Kernels erforderlich

## Einsatz von Crashdumps

- Postmortem-Analyse
- Kundensupport (→ »Enterprise«-Distributionen)
- Sporadische Systemabstürze
- Analyse von Lockups (System hängt)

# Geschichte der Crashdumps





## LKCD

- Extern gepflegter Kernelpatch: http://lkcd.sf.net
- Zusammenarbeit von SGI, HP, IBM
- Wurde mit SLES
- Dump über Netzwerk und Storage möglich
- Tool *lcrash* zum Analysieren der Dumps

## Netdump/Diskdump

- Alternativer Ansatz von Red Hat
- Heutige Dumpformate (ELF/makedumpfile) sind auf Basis der Formate von Netdump und Diskdump entstanden
- Tool *crash* hat »überlebt«

# Kexec





## Was ist *Kexec?*

- Mechanismus, um Linux aus Linux heraus zu booten
- Kein BIOS involviert

## Vor- und Nachteile

- *Pro:* kürzere Rebootzeiten, kein Bootloader nötig, keine Festplattenreihenfolge im BIOS
- *Contra:* (fehlerhafte) Gerätetreiber gehen von BIOS-Initialisierung aus

## Voraussetzungen

- Kernelkonfiguration: `CONFIG_KEXEC=y`
- Installation der *kexec-tools* von http://www.kernel.org/pub/linux/kernel/people/horms/kexec-tools/

# kdump





## Idee

- Nach Systemcrash wird mit *kexec* ein sog. *Capture-Kernel* ausgeführt
- Capture-Kernel erlaubt Zugriff auf alten Speicher und ein Programm sichert diesen Speicher

## Vorteile

- Zuverlässig: Kopieren erfolgt aus »sauberem« System
- Sichern des Dumps erfolgt im Userspace
  $\longrightarrow$ keine Netzwerkprotokoll-Implementierung im Kernel
- Normaler Hardwarezugriff möglich
  $\longrightarrow$ Zugriff auf Ethernet, SCSI, (S)ATA, USB, …

# Vier Schritte zum Dump

1. Speicher reservieren
2. Crashkernel laden mit `kexec`
3. System crashed
4. Dump sichern

# Kernel-Voraussetzungen





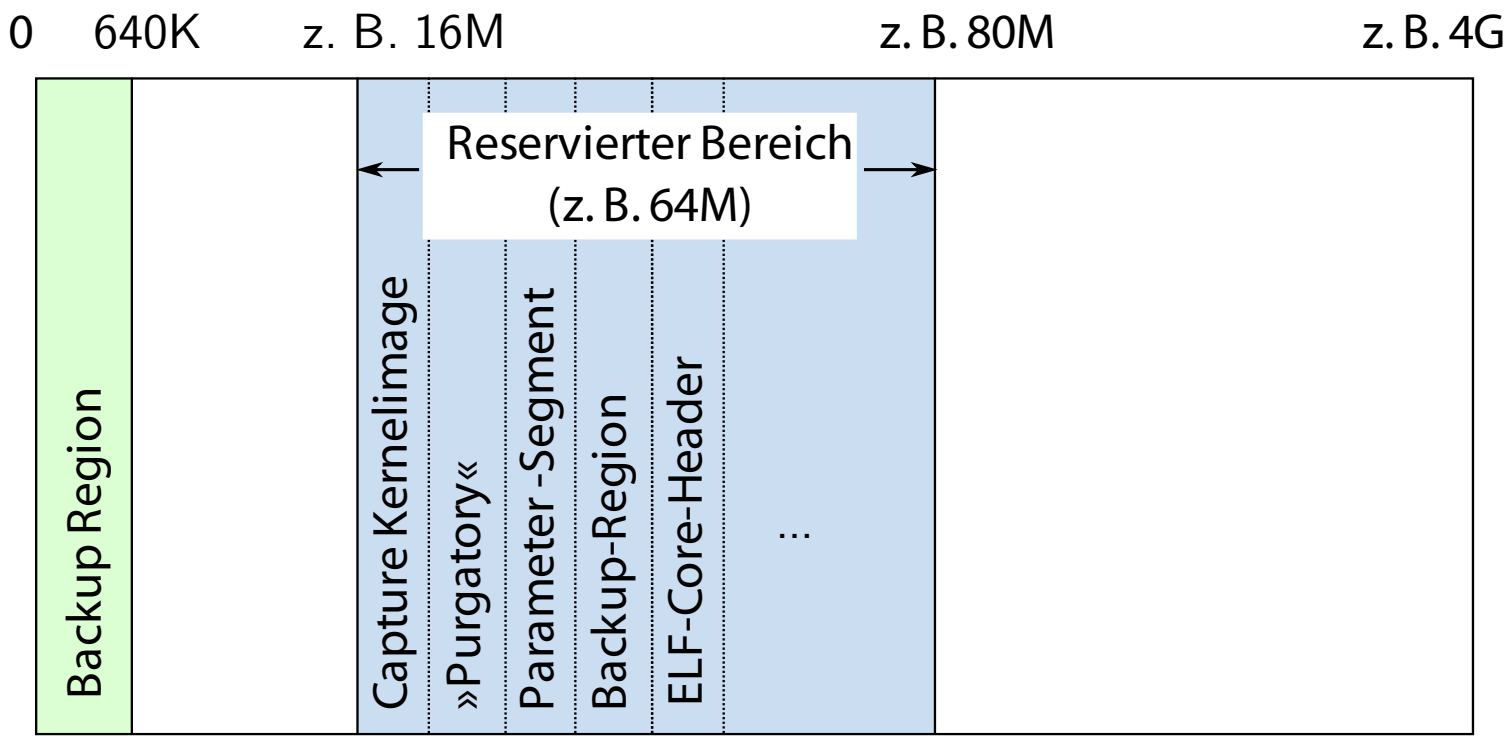
## Laufender Kernel

- `CONFIG_KEXEC=y`
- `CONFIG_DEBUG_INFO=y`

## Kernel, der den Dump sichern soll (»Panic-Kernel«)

- `CONFIG_CRASH_DUMP=y`
- `CONFIG_PROC_VMCORE=y` empfohlen
- `CONFIG_RELOCATABLE=y`
- Dieser Kernel kann auch als »normaler« Kernel verwendet werden, der Overhead ist vernachlässigbar!

# Warum Speicherreservierung?




- Kompletter »alter« Speicher soll im Panic-Kernel zur Verfügung stehen → Speicher darf im crashenden Kernel nicht genutzt werden
- x86: Speicher unter 640K wird zum Booten benötigt → Sichern erforderlich

# Speicherreservierung durchführen

- Kernelparameter `crashkernel`
- Format:

```
crashkernel=MENGE
```

- Ältere Kernelversionen

```
crashkernel=MENGE@STARTADRESSE
```

- Beispiel (Empfehlung für »normale« x86-Maschine):

```
crashkernel=64M
```

ℹ `Documentation/kdump/kdump.txt`

- Reservierung erfolgreich → `/proc/iomem`

```
00100000-bfedffff : System RAM
  01000000-0155ecc2 : Kernel code
  0155ecc3-019bce8f : Kernel data
  01abc000-01c4bf6f : Kernel bss
  02000000-09ffffff : Crash kernel <---- HIER
```

# Crashkernel laden

- Bei modernen Distributionen wird üblicherweise der gleiche Kernel verwendet!
- Crashkernel wird mit `kexec -p` (für »panic«) geladen

```
% kexec -p /boot/vmlinuz --initrd=/boot/initrd-kdump \
    --args="root=/dev/sda5 maxcpus=1 irqpoll \
            elevator=deadline reset_devices"
```

- Spezielle Kommandozeilenargumente
  - `maxcpus`: nur eine CPU im Panic-Kernel
  - `irqpoll`: alle IRQs beim Timerinterrupt abfragen
  - `elevator`: einfacher I/O-Scheduler
  - `reset_devices`: von wenigen Treibern implementiert
- In `/sys/kernel/kexec_crash_loaded` steht, ob gerade ein Crashkernel geladen ist
- Entladen wieder mit `kexec -p -u`

# Wann wird der Panic-Kernel gestartet?





## Automatisch

- Immer bei einer Kernelpanic
- x86: Panic bei NMI (Non-Masktable Interrupt) auslösen:

```
% sysctl kernel.panic_on_unrecovered_nmi=1
```

- Panic bei jedem Kernel-Oops auslösen:

```
% sysctl kernel.panic_on_oops=1
```

## kdump manuell testen

- `SysRq-S` (»emergency sync«)
- `SysRq-U` (»unmount readonly«)
- `SysRq-C` (»crash«)

# Dump sichern

## Panic-Environment

- normales System!
- langsamer (nur eine CPU, Interrupt-Polling)
- *deutlich* weniger Speicher
- evtl. keine VGA-Konsole → serielle Konsole
- sinnvoll: Dump in Initrd sichern

## Kopieren

- Zugriff auf alten Speicher
  - `/dev/oldmem` linear (wird praktisch nicht verwendet)
  - `/proc/vmcore` im ELF-Core-Format
- Sparse ausnutzen (falls Dateisystemunterstützung):

```
% cp --sparse=always /proc/vmcore /mnt/vmcore-20100313
```

# Dump filtern





## Motivation

- Dumpgröße $\approx$ Größe des Hauptspeichers
  $\longrightarrow$ bei großen Servern indiskutabel
- Kompression keine Lösung
  $\longrightarrow$ Vollständige Dekompression bei Analyse nötig

## Programm *makedumpfile*

- Filtern nach bestimmten Kriterien
  - Speicherseiten, die nur Nullen enthalten
  - Caches
  - Userspace-Daten
  - Freie Seiten (»Müll«)
- Seitenweise Kompression
  $\longrightarrow$ mit ELF nicht möglich, stattdessen Diskdump-Format

# Kdump bei SUSE, Red Hat und Ubuntu





## SUSE Linux Enterprise und openSUSE

- YaST-Modul: `yast2 kdump`
- Manuelle Konfiguration über `/etc/sysconfig/kdump`

ℹ *kdump(7)*

## Red Hat Enterprise und Fedora

- `system-config-kdump`
- Konfiguration über `/etc/kdump.conf`

ℹ http://kbase.redhat.com/faq/docs/DOC-6039

## Ubuntu

- `aptitude install linux-crashdump`

ℹ https://wiki.ubuntu.com/KernelTeam/CrashdumpRecipe

# *crash* starten





## Debuginformationen

- Eigener Kernel: `CONFIG_DEBUG_INFO`
- Distributionskernel: Debuginfo-Paket, z. B. `kernel-default-debuginfo`

## *crash*

- GDB-ähnliches Tool mit vielen kernelspezifischen Kommandos (enthält eingebauten GDB)
- unterstützt auch andere Dumpformate
- Aufruf:

```
% crash [Mapfile] Namelist Dumpfile
```

http://people.redhat.com/anderson/crash_whitepaper/

# *crash* starten (cont.)





## *crash*-Parameter

- **Mapfile:** die `System.map` (nur erforderlich wenn crashender Kernel nicht die `Namelist` ist)
- **Namelist:** `vmlinux` (ELF-Image) ist
- **Dumpfile:** Dump im ELF- oder `makedumpfile`-Format

## Empfehlungen

- Alles in separates Verzeichnis kopieren
  1. `vmcore`
  2. `System.map`
  3. `vmlinux`
  4. `vmlinux.debug`
- Emacs-Modus aktivieren (→ `$HOME/.crashrc`):

```
crash> set emacs
```

# Hilfe

```
crash> help

*               files           mod             runq            union
alias           foreach         mount           search          vm
ascii           fuser           net             set             vtop
bt              gdb             p               sig             waitq
btop            help            ps              struct          whatis
dev             irq             pte             swap            wr
dis             kmem            ptob            sym             q
eval            list            ptov            sys
exit            log             rd              task
extend          mach            repeat          timer

crash version: 5.0.0    gdb version: 7.0
For help on any command above, enter "help <command>".
For help on input options, enter "help input".
For help on output options, enter "help output".
```

```
crash> help KOMMANDO
```

# Systeminformationen anzeigen

```
crash> sys
      KERNEL: vmlinux
    DUMPFILE: vmcore
        CPUS: 2
        DATE: Mon Jan 11 22:24:01 2010
      UPTIME: 00:00:42
LOAD AVERAGE: 0.09, 0.03, 0.01
       TASKS: 149
    NODENAME: suse112
     RELEASE: 2.6.32.3
     VERSION: #4 SMP Mon Jan 11 22:04:42 CET 2010
     MACHINE: i686  (1937 Mhz)
      MEMORY: 255.5 MB
       PANIC: "[   42.735426] Oops: 0002 [#1] SMP " (check log fo
```

```
crash> sys config
#
# Automatically generated make config: don't edit
# Linux kernel version: 2.6.32.3
# Mon Jan 11 21:37:07 2010
#
# CONFIG_64BIT is not set
```

# Maschinenspezifische Systeminformationen

```
crash> mach
       MACHINE TYPE: i686
        MEMORY SIZE: 255.5 MB
               CPUS: 2
    PROCESSOR SPEED: 1937 Mhz
                 HZ: 1000
          PAGE SIZE: 4096
KERNEL VIRTUAL BASE: c0000000
KERNEL VMALLOC BASE: d07f0000
  KERNEL STACK SIZE: 8192
```

```
crash> mach -m
      PHYSICAL ADDRESS RANGE          TYPE
0000000000000000 - 00000000000a0000  E820_RAM
00000000000f0000 - 0000000000100000  E820_RESERVED
0000000000100000 - 000000001ff75000  E820_RAM
000000001ff75000 - 000000001ff77000  E820_NVS
000000001ff77000 - 000000001ff98000  E820_ACPI
000000001ff98000 - 0000000020000000  E820_RESERVED
00000000fec00000 - 00000000fec90000  E820_RESERVED
00000000fee00000 - 00000000fee10000  E820_RESERVED
00000000ffb00000 - 0000000100000000  E820_RESERVED
```

# Kernellog anzeigen

```
crash> log
Initializing cgroup subsys cpuset
Initializing cgroup subsys cpu
Linux version 2.6.32.3 (bwalle@suse112) (gcc version 4.4.1 [gcc-4_
KERNEL supported cpus:
  Intel GenuineIntel
  AMD AuthenticAMD
  NSC Geode by NSC
  Cyrix CyrixInstead
  Centaur CentaurHauls
  Transmeta GenuineTMx86
  Transmeta TransmetaCPU
  UMC UMC UMC UMC
BIOS-provided physical RAM map:
 BIOS-e820: 0000000000000000 - 000000000009fc00 (usable)
 BIOS-e820: 000000000009fc00 - 00000000000a0000 (reserved)
 BIOS-e820: 00000000000f0000 - 0000000000100000 (reserved)
 BIOS-e820: 0000000000100000 - 000000000fff0000 (usable)
 BIOS-e820: 000000000fff0000 - 0000000010000000 (ACPI data)
 BIOS-e820: 00000000fffc0000 - 0000000100000000 (reserved)
DMI 2.5 present.
last_pfn = 0xfff0 max_arch_pfn = 0x100000
MTRR default type: uncachable
```

# Backtrace (aktueller Task)

```
crash> bt
PID: 3047   TASK: cf847860  CPU: 0   COMMAND: "bash"
 #0 [cee05ddc] crash_kexec at c105c0c3
 #1 [cee05e30] oops_end at c1485d1c
 #2 [cee05e44] no_context at c101b036
 #3 [cee05e6c] __bad_area_nosemaphore at c101b120
 #4 [cee05e84] __bad_area at c101b16b
 #5 [cee05e9c] bad_area at c101b17e
 #6 [cee05ea8] do_page_fault at c1486d73
 #7 [cee05ed4] error_code (via page_fault) at c14854c4
    EAX: 00000063  EBX: 00000063  ECX: 00200046  EDX: 00000000  EB
    DS:  007b      ESI: c16e14b8  ES:  007b      EDI: 00000007  GS
    CS:  0060      EIP: c1214609  ERR: ffffffff  EFLAGS: 00210046
 #8 [cee05f08] sysrq_handle_crash at c1214609
 #9 [cee05f18] __handle_sysrq at c1214786
#10 [cee05f40] write_sysrq_trigger at c1214828
#11 [cee05f50] proc_reg_write at c10e04e3
#12 [cee05f74] vfs_write at c10ac3c2
#13 [cee05f90] sys_write at c10ac4b2
#14 [cee05fb0] ia32_sysenter_target at c100299d
    EAX: 00000004  EBX: 00000001  ECX: b744e000  EDX: 00000002
    DS:  007b      ESI: 00000002  ES:  007b      EDI: b744e000
    SS:  007b      ESP: bf834728  EBP: b771f4e0  GS:  0000
    CS:  0073      EIP: b776e424  ERR: 00000004  EFLAGS: 00200246
```

# Laufende Prozesse

```
crash> ps
   PID    PPID  CPU   TASK    ST  %MEM     VSZ    RSS  COMM
     0       0   0  c16c4e60  RU   0.0       0      0  [swapper]
>    0       0   1  cf845ec0  RU   0.0       0      0  [swapper]
     1       0   0  cf844000  IN   0.3    1944    656  init
     2       0   0  cf844520  IN   0.0       0      0  [kthreadd]
     3       2   0  cf844a40  IN   0.0       0      0  [migration
     4       2   0  cf844f60  IN   0.0       0      0  [ksoftirqd
     5       2   1  cf845480  IN   0.0       0      0  [migration
     6       2   1  cf8459a0  IN   0.0       0      0  [ksoftirqd
     7       2   0  cf8463e0  IN   0.0       0      0  [events/0]
     8       2   1  cf846900  IN   0.0       0      0  [events/1]
     9       2   0  cf846e20  IN   0.0       0      0  [cpuset]
    10       2   0  cf847340  IN   0.0       0      0  [khelper]
    13       2   0  cf88c520  IN   0.0       0      0  [netns]
    16       2   1  cf88d480  IN   0.0       0      0  [async/mgr]
   230       2   0  cf88dec0  IN   0.0       0      0  [sync_super
   232       2   0  cf88e900  IN   0.0       0      0  [bdi-defaul
   234       2   0  cf88e3e0  IN   0.0       0      0  [kblockd/0]
   235       2   1  cf88d9a0  IN   0.0       0      0  [kblockd/1]
   237       2   0  cf88ca40  IN   0.0       0      0  [kacpid]
   238       2   0  cf88cf60  IN   0.0       0      0  [kacpi_noti
   239       2   0  cf88f860  IN   0.0       0      0  [kacpi_hotp
```

# Simple Image Access Language (SIAL)

http://people.redhat.com/anderson/extensions.html

```
crash> extend sial.so
Core LINUX_RELEASE == '2.6.32.3'
< Sial interpreter version 3.0 >
        Loading sial commands from /usr/share/sial/crash:/home/bwa
/usr/lib/crash/extensions/sial.so: shared object loaded
```

```
crash> load ps.c # $HOME/.sial und /usr/share/sial/crash
```

```
crash> help sps

NAME
  sps -

SYNOPSIS
  sps [-l] [-t] [-h]

DESCRIPTION
  This command displays various information about processes.
```

# Inhalt

# Interaktive Kernel-Debugger





## Motivation

- Kernelprogramme genauso wie Userspace-Programme debuggen

## Historie

- Linus war lange gegen interaktive Debugger, da sie seiner Meinung nach die Qualität des Kernels verschlechtern
  $\longrightarrow$ Man soll den Code verstehen!
- KDB als externer Patch
- KGDB als externer Patch (lange Zeit im »mm-Tree«)
- Ingo Molnar entwickelt abgespeckten KGDB
- Aufnahme des KGDB in den Linux-Kernel

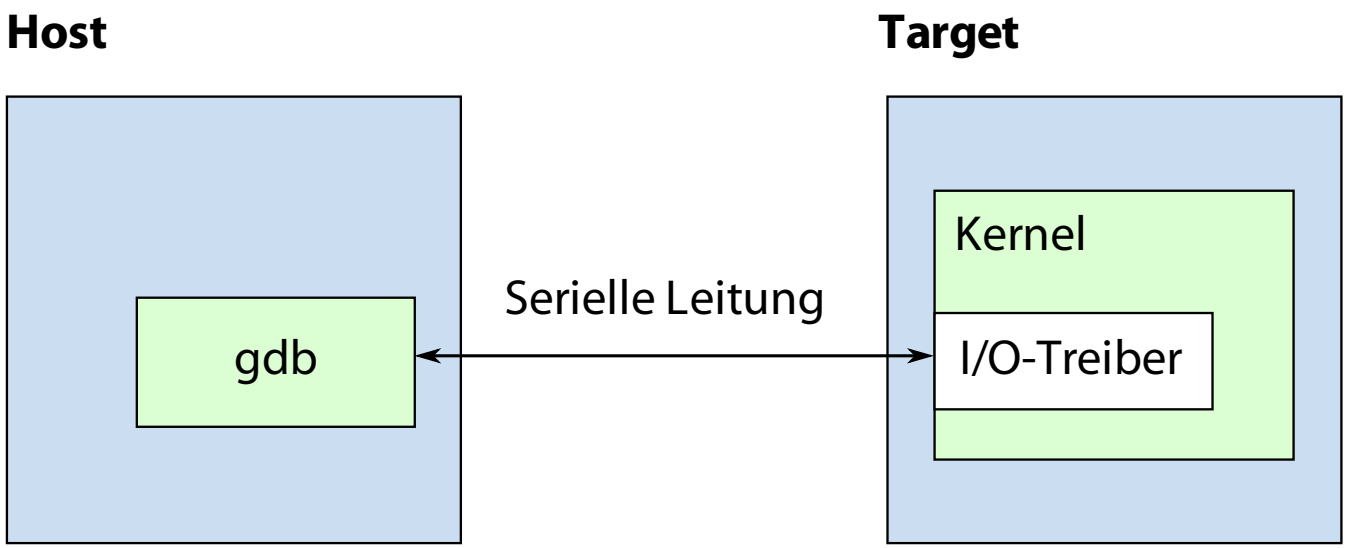
# KDB

- Externer Kernelpatch
- Quelle: http://oss.sgi.com/projects/kdb/
  (wird sehr schnell an neue Kernel angepasst)
- VGA-Konsole und serielle Konsole
- Unterstützt USB-Tastaturen
- Architekturen: x86 (32 und 64 Bit) und IA64
- `PAUSE`-Taste springt in KDB
- kein Debuggen auf Quellcodebene

# KGDB








- Prinzip: Remote Debugging mit GDB
- Im offiziellen Kernel enthalten

`Documentation/DocBook/kgdb.tmpl`

# Konfiguration





## Kernel

- `CONFIG_KGDB=y`
- `CONFIG_FRAME_POINTER=y` (falls verfügbar)
- `CONFIG_DEBUG_RODATA=n` (für Breakpoints)
- `CONFIG_KGDB_SERIAL_CONSOLE=(y|n)`
- `CONFIG_DEBUG_INFO=y`

## KGDB als Modul

- Modul laden

```
# modprobe kgdboc kgdboc=/dev/ttyS0,115200
```

- KGDB starten (stoppt Kernel!)
  → `SysRq-G`

# Konfiguration (cont.)





## KGDB einkompiliert

- Kernelkommandozeile

```
kgdboc=/dev/ttyS0,115200
```

- KGDB wie oben starten

## KGDB direkt beim Booten starten

- Kernelkommandozeile

```
kgdboc=/dev/ttyS0,115200 kgdbwait
```

## Konfiguration nachträglich ändern

- Während KGDB nicht läuft

```
% echo ttyS1 > /sys/module/kgdboc/parameters/kgdboc
```

# KGDB nutzen





## Voraussetzung

Kernel muss auf Verbindung warten

## GDB starten

```
% gdb vmlinux
(gdb) set remotebaud 115200
(gdb) target remote /dev/ttyUSB0
```

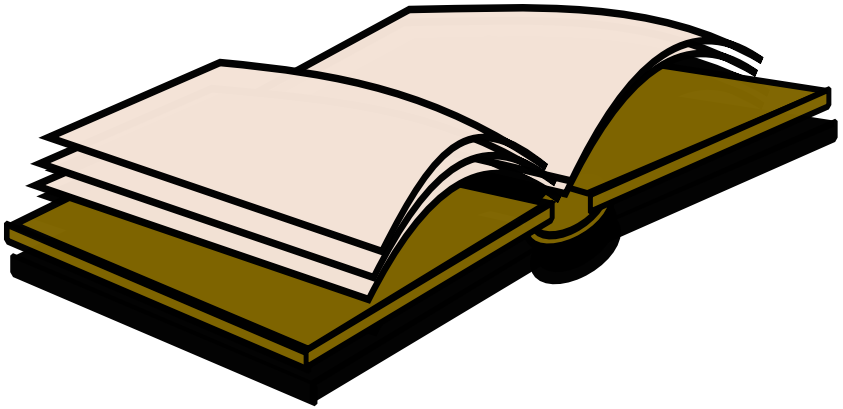
## Breakpoint setzen

```
% gdb vmlinux
(gdb) break crash_kexec
(gdb) continue
```

## KGDB reagiert nicht (Debugging)

```
(gdb) set debug remote 1
```

# Vortrag nachlesen



http://www.bwalle.de/website/clt2010.html

# Literatur zur Kernel-Konsole und Kernel-Debugging





Greg Kroah-Hartman
*Linux Kernel in a Nutshell*
O'Reilly-Verlag

Jürgen Quade, Eva-Katharina Kunst
*Linux-Treiber entwickeln*
dpunkt-Verlag

Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman
*Linux Device Drivers*
O'Reilly-Verlag

Glen Turner, Mark F. Komarinski
*Remote Serial Console HOWTO*
The Linux Documentation Project

# Literatur zu Crashdumps





Vivek Goyal, Eric W. Biederman, Hariprasad Nellitheertha
*Kdump, A Kexec-based Kernel Crash Dumping Mechanism*
OLS 2005

Vivek Goyal, Neil Horman, Ken'ichi Ohmichi, Maneesh Soni, Ankita Garg
*Kdump: Smarter, Easier, Trustier*
OLS 2007

David Anderson
*White Paper: Red Hat Crash Utility*
Red Hat Software, Inc.