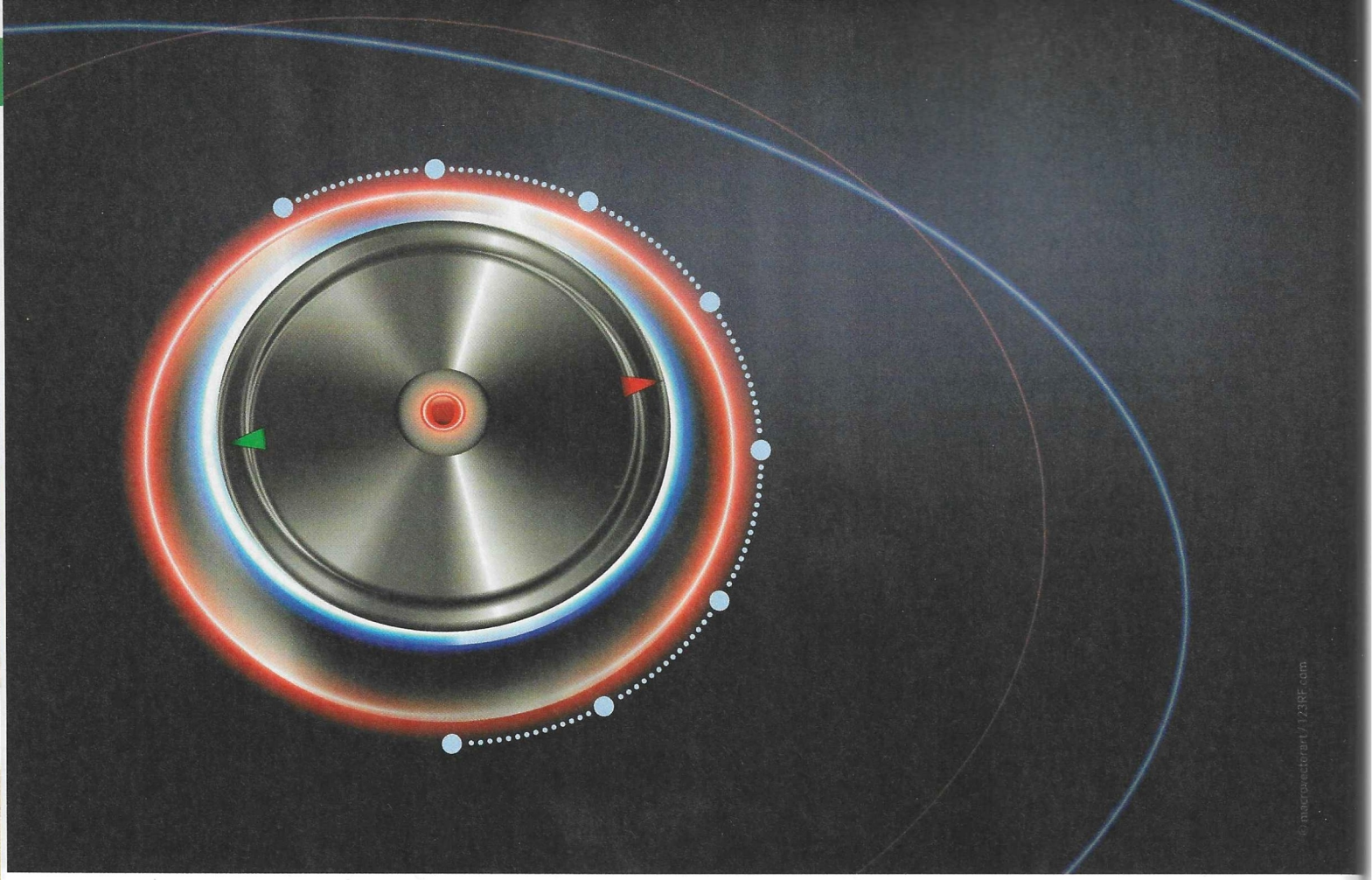


Kernel- und Treiberprogrammierung mit dem Linux-Kernel – Folge 129

Lautstärkeregler

Der Linux-Kernel unterstützt pfiffig sowohl Admins beim Umgang mit Kernel-Nachrichten als auch Entwicklerinnen und Entwickler bei deren Generierung. Eva-Katharina Kunst, Jürgen Quade

Programmierkarrieren beginnen typischerweise mit einem „hello, world". Dessen Ursprung geht auf die beiden AT&T-Mitarbeiter Brian Kernighan und Dennis Ritchie zurück, die die Programmiersprache C erfunden haben. Mit `printf("hello, world");` gaben sie 1974 als Erste den berühmt gewordenen String auf die Konsole aus. Dabei ist `printf()` nicht einmal Teil der Programmiersprache C selbst, sondern Teil der C-Standardbibliothek.

Auch ein Linux-Kernel hat Mitteilungsdrang, und da der Betriebssystemkern keinen Zugriff auf die Standardfunktionen der C-Bibliothek hat, spendierte ihm sein Vater Linus Torvalds konsequenterweise eine korrespondierende Funktion `printk()`. In den ersten Versionen war dieses Sprachrohr des Kernels im Vergleich zu `printf()` deutlich abgespeckt und unterstützte nur wenige, ausgewählte Umsetzungssequenzen. Heute ist die Funktion stark ausgeweitet und vor allem auch auf spezifische Belange eines Betriebssystemkerns hin ausgerichtet. Dadurch ist der Umgang mit der Funktion teilweise etwas kniffelig geworden.

Geplagte Admins

Per `printk()` publizierte Kernel-Nachrichten werden nicht einfach blind auf die armen Admin-Profis losgelassen. Vielmehr bietet der Kernel der geplagten Systemverwaltung die Möglichkeit, die Flut an Ausgaben aus dem Inneren des Betriebssystemkerns per Konfiguration einzudämmen. Dazu dienen der System

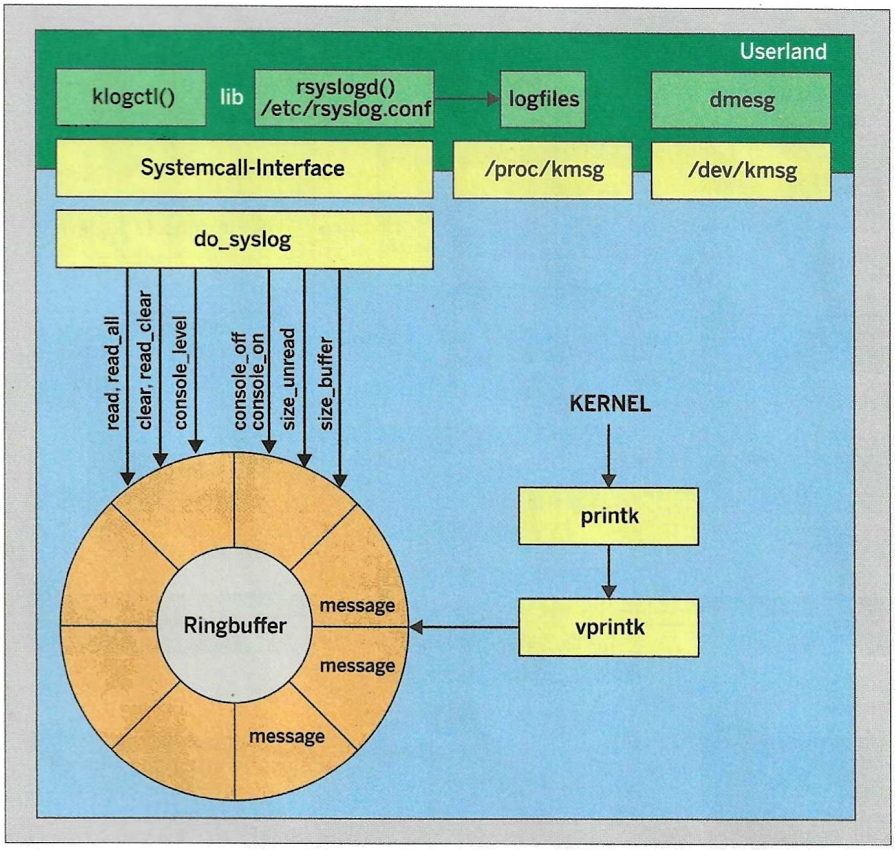
Die Autoren

Jürgen Quade, Professor an der Hochschule Niederrhein, gibt auch für Unternehmen Schulungen zu den Themen Treiberprogrammierung und Embedded Linux. Eva-Katharina Kunst ist seit den Anfängen von Linux Fan von Open Source.

Call `syslog()`, den die Standard-C-Bibliothek auf die Funktion `klogctl()` abbildet, sowie der im Userland laufende Syslog-Daemon (`rsyslogd()`) 1. Hinzu kommen eine Reihe von virtuellen Dateien im Verzeichnis `proc/`.

Insbesondere konfiguriert man mit dem System Call `syslog()` den Konsolen-Level. Dazu sollte man wissen, dass alle Nachrichten eines Betriebssystemkerns klassischerweise auf der sogenannten Konsole landen. In den Anfangstagen von Unix hatte jeder Rechner ein über ein serielles Interface angeschlossenes Terminal, die Konsole. Heute, in Zeiten grafischer Displays, ist die Konsole zumeist virtuell. Am ehesten bekommt man sie beim Booten zu Gesicht; insbesondere, wenn man während des Boot-Vorgangs durch einen Druck auf [Esc] das grafische Boot-Logo gegen die Konsolennachrichten tauscht.

Um die Konsole während des Betriebs nicht vollzuspammen, kennzeichnet der Kernel alle seine Nachrichten mit einem Log-Level zwischen 0 und 7 (siehe Tabelle Log-Level und deren Abbildung im Kernel). Diese Log-Level entsprechen denen des Syslog-Systems. Per System Call lässt sich die Schwelle konfigurieren, ab der die Mitteilungen ausgegeben werden. Dabei entspricht eine niedrige Zahl einer hohen Priorität.



1 **Die Architektur des Kernel-Log-Systems im Überblick.**

Plappermäuler stopfen

Bei Bedarf stellen Sie die Schwelle über das Proc-Filesystem beziehungsweise mithilfe von `sysctl` und der zugehörigen Konfigurationsdatei `/etc/sysctl.conf` ein 2. Auf die Proc-Datei `/proc/sys/kernel/printk` setzen Sie durch vier Ganzzahlen vier Attribute. Die erste Ganzzahl repräsentiert die erwähnte Schwelle. Nur höher als die Schwelle priorisierte Nachrichten, also solche mit einem kleineren Log-Level, werden protokolliert. Die zweite Ganzzahl spezifiziert den Default-Log-Level, falls eine Nachricht keine Priorisierung mitbringt.

Als Nächstes folgt der `minimum_console_loglevel`. Er sichert die Konfiguration des Default-Log-Levels ab, indem er eine Untergrenze dafür angibt. Damit lassen sich Bösewichter ausbremsen, die durch geschickte Änderung des Log-Levels ihr Tun verbergen wollen. Der vierte Wert gibt den Standard für den Log-Level vor. Auf dieser Priorisierungsebene werden Nachrichten verarbeitet, die keine Log-Level-Markierung aufweisen.

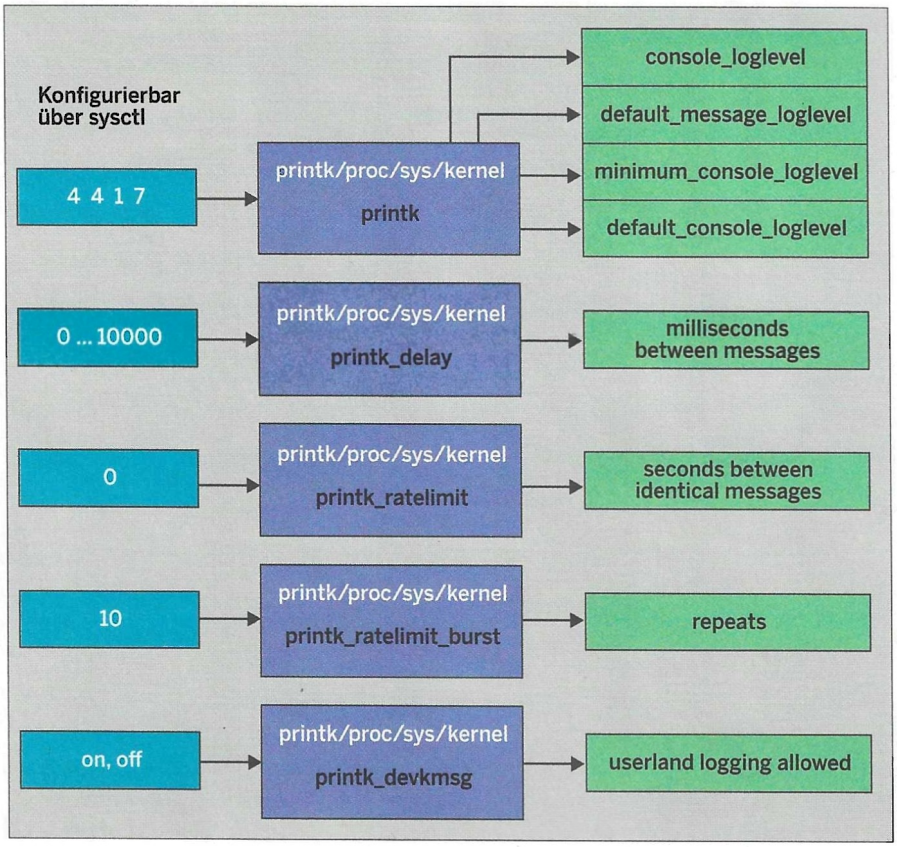
Über die Proc-Datei `/proc/sys/kernel/printk_delay` lassen sich Kernel-Meldungen um die dort konfigurierte Zeit verzögern. Erlaubt sind Werte zwischen 0 Millisekunden (keine Verzögerung) und 10 000 Millisekunden, also 10 Sekunden.

Ein häufiges Problem stellen wiederholt auftretende, identische Meldungen dar. Den zeitlichen Abstand zwischen zwei Meldungen stellen Sie über `printk_ratelimit` ein (in Sekunden) beziehungsweise lesen ihn aus. Über `printk_ratelimit_burst` legen Sie eine Anzahl von Wiederholungen fest, bevor die Zeitbedingung greift. Standardmäßig sind zehn Wiederholungen konfiguriert.

Über die Proc-Datei `printk_devkmsg` konfigurieren Sie, ob Applikationen über die Gerätedatei `/dev/kmsg` eigene Meldungen in das Kernel-Logging einpflegen dürfen. Standardmäßig ist das er-

Log-Level und deren Abbildung im Kernel

Nummer	Syslog-Name	Define	Ausgabefunktion
0	Emergency	KERN_EMERG	pr_emerg()
1	Alert	KERN_ALERT	pr_alert()
2	Critical	KERN_CRIT	pr_crit()
3	Error	KERN_ERR	pr_err()
4	Warning	KERN_WARNING	pr_warning()
5	Notice	KERN_NOTICE	pr_notice()
6	Informational	KERN_INFO	pr_info()
7	Debug	KERN_DEBUG	pr_debug(), pr_devel()



2 Per `sysctl` lässt sich das Log-System konfigurieren.

laubt, sicherheitsbewusste Admins stellen den Wert aber gern auf `off`.

Sie sollten im Hinterkopf behalten, dass der Kernel bei Aufruf von `printk()` die Nachrichten mit einem Zeitstempel versehen in einen Ringpuffer ablegt. Dessen Größe lässt sich konfigurieren, unter Ubuntu liegt sie bei 256 MByte. Diese Größe sowie die Frequenz, mit der neue Nachrichten auftreten und aus dem Ringpuffer gelesen werden, entscheiden darüber, ob Informationen verloren gehen. Die im Ringpuffer abgelegten Nachrichten lassen sich über den System Call `syslog` respektive via `klogctl` auslesen und als wieder überschreibbar markieren (siehe Tabelle „Kommandos der Funktion `klogctl()` (Auswahl)"). Alternativ kann der Zugriff über die Gerätedatei `/dev/kmsg` erfolgen. Zudem existiert ein Zugang über `/proc/kmsg`, der nur die neu hinzugekommenen Einträge liest.

Kommandos der Funktion `klogctl()` (Auswahl)

Kommando	Wert	Bedeutung
SYSLOG_ACTION_READ	2	Auslesen einer Nachricht
SYSLOG_ACTION_READ_ALL	3	Auslesen aller Nachrichten
SYSLOG_ACTION_READ_CLEAR	4	Lesen und Freigeben aller Nachrichten
SYSLOG_ACTION_CLEAR	5	Freigeben zum Überschreiben der Nachrichten
SYSLOG_ACTION_CONSOLE_OFF	6	Konsolenausgaben stoppen
SYSLOG_ACTION_CONSOLE_ON	7	Konsolenausgaben freigeben
SYSLOG_ACTION_CONSOLE_LEVEL	8	Log-Level-Schranke festlegen
SYSLOG_ACTION_SIZE_UNREAD	9	Anzahl Bytes ungelesener Nachrichten
SYSLOG_ACTION_SIZE_BUFFER	10	Auslesen der Log-Level-Puffergröße

Das den meisten Anwendern wohlvertraute Kommando `dmesg` greift übrigens auf `/dev/kmsg` zurück, um den Inhalt des Ringpuffers auszulesen und auf dem Terminal zu publizieren. Der schreibende Zugriff auf `/dev/kmsg` ermöglicht es, eigene Einträge in den Ringpuffer abzulegen.

Kernel-Lektüre

Auf den meisten Linux-Systemen gewährleisten der Rsyslog-Daemon oder Alternativen wie `syslog-ng` das regelmäßige Auslesen. Der Dienst ist grundsätzlich für das Event-Management eines Linux-Systems zuständig. Dank der umfangreichen Konfigurationsmöglichkeiten leitet Rsyslogd die Nachrichten nicht nur in Log-Dateien um, sondern protokolliert auch per Netzwerk auf andere Server.

Die Filtermöglichkeiten für jedes Log-Level müssen sich im Kernel im Aufruf von `printk()` widerspiegeln. Linus Torvalds hat sich dazu eine Lösung ausgedacht, die weitgehend kompatibel zu `printf()` arbeitet. Statt einen zusätzlichen Parameter für den Log-Level einzuführen, lässt sich dieser optional mit einer Ganzzahl am Anfang des sogenannten Format-Strings durch Setzen in spitzen Klammern spezifizieren.

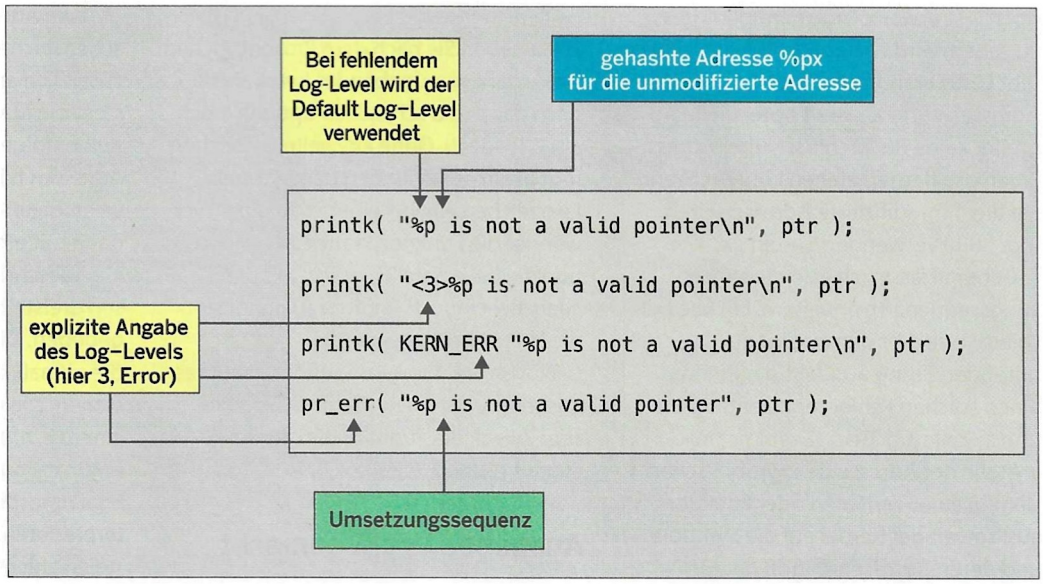
Um den Code etwas lesbarer zu gestalten, spaltet der Linux-Kernel den Format-String in diesen Anfangsteil und die eigentliche Ausgabe auf. Für den Anfangsteil, also die Priorität als Ganzzahl in spitzen Klammern, hat Torvalds Defines festgelegt. Das führt zu einem Aussehen, das beispielsweise in Zeile 11 in Listing 1 zu sehen ist. Man beachte, dass zwischen dem Define und dem restlichen Format-String kein Komma stehen darf.

Zusätzlich gibt es noch Makros wie `pr_info()` (Zeile 12) oder `pr_debug()` (Zeile 13). Die mit `pr_` beginnenden Makros ergänzen den Format-String um den zugehörigen Log-Level sowie zusätzlich um den Modulnamen. In Abgrenzung zu `pr_info()` wird `pr_debug()` nur dann mitkompiliert, wenn das Define `KERN_DEBUG` gesetzt ist. In einem Gerätetreiber empfehlen sich im Übrigen Varianten, die mit `dev_` beginnen: Sie ergänzen über einen zusätzlichen Parameter die Ausgabe um das Gerät (Device).

Apropos Debug: In der Trickkiste von Linux befinden sich noch die Makros

`bug()`, `bug_on()`, `warn()`, `warn_on()` und `warn_once`. Allen diesen Makros übergeben Sie als ersten Parameter eine Bedingung. Trifft sie zu, geben die Routinen den Stacktrace aus. Aus ihm lässt sich die Aufrufkaskade ablesen, was sehr nützlich zur Fehleranalyse ist.

Die `warn()`-Makros begnügen sich mit einer umfangreichen Ausgabe, die `bug()`-Routinen dagegen lösen eine Kernel Panic und das Beenden des aktiven Kernel-Codes aus. Während die einfachen Spielarten mit dem Stacktrace den Namen des Quellcodemoduls und die Zeilennummer ausgeben, bekommen die _on-Varianten einen Format-String samt potenziellen Parametern überreicht. Das Makro `warn_once` stellt sicher, dass die Meldung nur ein einziges Mal ausgegeben wird und nicht gleich das ganze System flutet.



3 **Diverse `printk()`-Varianten erhöhen die Lesbarkeit.**

Tückisches `printk()`

Nicht nur seine Syslog-Anbindung zeichnet `printk()` aus, sondern auch die umfangreichen Formatierungsmöglichkeiten. Jedes Prozentzeichen im Format-String leitet eine sogenannte Umsetzungssequenz ein. Sie wird bei der Ausgabe durch einen Wert oder Text ersetzt, den `printk()` als weiteren Parameter übergeben bekommt.

Die durch das Prozentzeichen eingeleitete Umsetzungssequenz bestimmt durch zusätzliche, angehängte Zeichen, wie es den Wert zu formatieren gilt. So sorgt ein `%d` für eine dezimale Ausgabe des übergebenen Werts, ein `%x` präsentiert ihn in seiner hexadezimalen Form. Die Umsetzungssequenzen für die Standarddatentypen sind mit denen von `printf()` identisch. Umsetzungssequenzen für Fließkommazahlen fehlen jedoch, da es die im Kernel nicht gibt. Auch das sicherheitsgefährdende `%n` wurde glücklicherweise aus dem Betriebssystemkern verbannt.

Exotisch und manchmal tückisch wird es bei der Ausgabe von Adressen, also Pointern. Das klassische `%p` existiert zwar,

Listing 1: `printktest.c()`

```
01 #include <linux/module.h>
02 #include <linux/cdev.h>
03
04 static int __init mod_init(void) {
05   char *ptr = NULL;
06   char field[64];
07   __be32 ipv4addr = 0x2a2aa8c0;
08
09   printk("mod_init called\n");
10   ptr = (char *)-0x1;
11   printk( KERN_INFO "Pointer (gehasht und
     orginal)): %p - %px\n", field, field );
12   pr_info("Pointer (Fehlertext und Wert): %pe -
     %p\n", ptr, ptr );
13   pr_debug("Pointer (symbolisch): %pS - %pS\n",
     mod_init, cdev_add );
14   pr_err("Pointer (symbolisch, Modul): %pSb -
     %pSb\n", mod_init, cdev_add);
15   printk("Feld (8 Elemente): %8ph\n", field );
16   printk("IPv4-Adresse: %pI4\n", &ipv4addr );
17   print_hex_dump( KERN_INFO, "Linux-Mag: ", DUMP_
     PREFIX_NONE,16,1,field,sizeof(field),1);
18   // WARN( ptr==(char*)(-1), "Uppps, just for
     fun");
19   return -1;
20 }
21
22 static void __exit mod_exit(void)
23 {
24     printk("mod_exit called\n");
25 }
26 module_init( mod_init );
27 module_exit( mod_exit );
28 MODULE_LICENSE( "GPL" );
```

doch wer erwartet, die richtige Adresse zu sehen, wird enttäuscht 3. Tatsächlich gibt Linux standardmäßig nur gehashte Adressen zurück, damit potenzielle Angreifer keine Rückschlüsse über Adresslagen des Kernels ziehen können. Wenn Sie die unmodifizierte Adresse sehen möchten, verwenden Sie ein `px`.

Generell lässt sich also die Umsetzungssequenz `%p` erweitern. Ein `%pe` beispielsweise gibt den zu einem Fehler gehörenden String aus, falls der Pointer einen solchen Fehler repräsentiert (Listing 1, Zeile 13). Bei `%pS` gibt `printk()` anstelle der Adresse den symbolischen Namen einer Variablen oder Funktion aus, soweit der Kernel auf die Symbole und deren Adressen Zugriff hat. Bei einem weiter angehängtem `b`, also `%pSb`, erscheint zusätzlich der Modulname.

Die Funktion `printk()` kann darüber hinaus Felder bis zu einer Größe von 64 Byte in hexadezimaler Form liefern. Dazu geben Sie nach dem Prozentzeichen die Anzahl auszugebender Bytes an, danach das `p` und schließlich optional noch ein `C`, `D` oder `N`. Ohne ein weiteres Zeichen fügt `printk()` zwischen den Bytes ein Leerzeichen ein. Bei einem `C` („colon") verwendet die Funktion stattdessen Doppelpunkte, bei einem `D` („dash") Minuszeichen. Bei einem `N` („nothing") hängt sie die Bytes direkt aneinander.

Wollen Sie übrigens mehr als 64 Bytes ausgeben, dann empfiehlt sich zu diesem Zweck die Funktion `print_hex_dump()` (Listing 1, Zeile 17).

Ausgaben leicht gemacht

Die Kernel-Funktion `printk()` kann noch mehr. So gibt es Umsetzungssequenzen für MAC- und IP-Adressen. Dazu dient ein dem `p` angehängtes `M` oder `m`. Ersteres gibt die Adressen inklusive eines Zwischenzeichens aus, Letzteres ohne. Ein zusätzlich angehängtes `R` („reverse") stellt die MAC-Adresse in umgekehrter Reihenfolge dar, wie es bei Bluetooth-Adressen häufig üblich ist.

Für die IP-Adressen kombinieren Sie das `p` mit einem `I4` (Zeile 16). Bei einem kleingeschriebenen `i` werden die Adressen dreistellig mit führenden Nullen ausgegeben. Die `4` steht für IPv4-Adressen, die gemäß der Norm aus vier Bytes bestehen. Die `4` lässt sich übrigens noch mit einem `h`, `n`, `b` oder `l` kombinieren, um sie im Host-Format, im Netzformat, als Big-Endian respektive als Little-Endian zu interpretieren.

Es ist nicht schwer zu erraten, dass der Kernel auch IPv6 unterstützt und Sie dazu das `p` mit einem `I6` oder `i6` kombinieren. Das kleine `i` lässt wieder die Zwischenzeichen weg. Ein zusätzlich angehängtes `c` stellt die IPv6-Adresse in Kurzform dar, bei der beispielsweise Folgen von Nullen bei Eindeutigkeit wegfallen dürfen.

Die Tabelle Ausgewählte Umsetzungssequenzen listet Umsetzungssequenzen für weitere spezifische Datentypen auf, so etwa 16 Byte breite UUIDs und GUIDs, Dentry-Namen, Blockgerätenamen oder Devicetree-Knoten. Auch Fwnodes und Zeitstempel lassen sich automatisiert und standardisiert ausgeben. Daneben finden sich Clocks und Bitmaps ebenso im Programm wie Flags, die bei Seitentabellen vorkommen. Eine Auflistung finden Sie auf Kernel.org.

Sofern der Quellcode aus Listing 1 in der Datei `printktest.c` vorliegt, es ein passendes Makefile gibt und Sie die geeigneten Pakete zur Generierung von Kernel-Code (*build-essential*, *flex*, *bison*) installiert haben, können Sie unser Beispiel durch Aufruf von `make` zum Modul `printktest.ko` generieren lassen.

Ausgewählte Umsetzungssequenzen

Sequenz	Formatierung
`%p`	gehashte Adresse
`%px`	unmodifizierte Adresse
`%pe`	Fehlertext
`%pS`	Adresse als Symbol mit Offset
`%ps`	Adresse als Symbol ohne Offset
`%pK`	Kernel-Pointer (Sichtbarkeit konfigurierbar)
`%td`	Differenz zweier Pointer (dezimal)
`%tx`	Differenz zweier Pointer (hexadezimal)
`%pr`	`struct resources` (ohne Dekodierung der Flags)
`%pR`	`struct resources` (mit Dekodierung der Flags)
`%pap`	`hphys_addr_t`
`%pad`	`dma_addr_t`
`%*pE`	Raw-Buffer als Escape-Sequenz
`%*ph`	Raw-Buffer als Hex-String
`%pM`	MAC-Adresse
`%pI4`	IPv4-Adresse
`%pi4`	IPv4-Adresse mit führenden Nullen
`%pI6`	IPv6-Adresse
`%pub`	16-Byte-UUID/GUIDs Adressen (Big-Endian)
`%pul`	16-Byte-UUID/GUIDs Adressen (Little-Endian)
`%pOF`	Devicetree-Knoten
`%ptR`	`struct rtc_time`
`%ptT`	`time64_t`
`%pC`	`struct clk`
`%*pb`	Bitmaps

Dateien zum Artikel herunterladen unter
www.lm-online.de/dl/48906

Weitere Infos und interessante Links
www.lm-online.de/qr/48906

Das Kommando `insmod printktest.ko` lädt das Modul, wobei das `return -1` am Ende der Funktion `mod_init()` ein Einnisten in den Kernel verhindert. Das erklärt auch die (letztlich beabsichtigte) Fehlermeldung beim Laden.

Um sich die Ausgabe anzeigen zu lassen, geben Sie beispielsweise das Kommando `tail -n 10 /var/log/kern.log` auf der Konsole ein. Abbildung 4 zeigt das Makefile, die Generierung, das Laden des Moduls und schließlich die Ausgabe über die Datei `/var/log/kern.log`. Dabei erkennen Sie die unterschiedlichen Formatierungen von Pointern.

```
quade@ezs-cocka:~/linux-magazin$ cat Makefile
obj-m += printktest.o

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
quade@ezs-cocka:~/linux-magazin$ make
make -C /lib/modules/5.15.0-75-generic/build M=/home/quade/linux-magazin modules
make[1]: Verzeichnis „/usr/src/linux-headers-5.15.0-75-generic" wird betreten
  CC [M]  /home/quade/linux-magazin/printktest.o
  MODPOST /home/quade/linux-magazin/Module.symvers
  CC [M]  /home/quade/linux-magazin/printktest.mod.o
  LD [M]  /home/quade/linux-magazin/printktest.ko
  BTF [M] /home/quade/linux-magazin/printktest.ko
Skipping BTF generation for /home/quade/linux-magazin/printktest.ko due to unavailability of vmlinux
make[1]: Verzeichnis „/usr/src/linux-headers-5.15.0-75-generic" wird verlassen
quade@ezs-cocka:~/linux-magazin$ sudo insmod printktest.ko
insmod: ERROR: could not insert module printktest.ko: Operation not permitted
quade@ezs-cocka:~/linux-magazin$ tail -n 10 /var/log/kern.log
Jul  2 14:25:27 ezs-cocka kernel: [5191044.437201] printktest started...
Jul  2 14:25:27 ezs-cocka kernel: [5191044.437208] Pointer (gehasht und orginal)): 00000000302176fb - ffffbc8bc5eff850
Jul  2 14:25:27 ezs-cocka kernel: [5191044.437216] Pointer (Fehlertext und Wert): -EPERM - ffffffffffffffff
Jul  2 14:25:27 ezs-cocka kernel: [5191044.437221] Pointer (symbolisch, Modul): mod_init+0x0/0x1000 [printktest] - cdev_add+0x0/0x70
Jul  2 14:25:27 ezs-cocka kernel: [5191044.437240] Feld (3 Elemente): 00 00 00 00 00 00 00 00
Jul  2 14:25:27 ezs-cocka kernel: [5191044.437255] IPv4-Adresse: 192.168.42.42
Jul  2 14:25:27 ezs-cocka kernel: [5191044.437260] Linux-Mag: 00 00 00 00 00 00 00 00 53 82 00 00 00 00 00 00  ........S.......
Jul  2 14:25:27 ezs-cocka kernel: [5191044.437266] Linux-Mag: 98 fb ef c5 8b bc ff ff fa 5a 71 99 ff ff ff ff  .........Zq.....
Jul  2 14:25:27 ezs-cocka kernel: [5191044.437272] Linux-Mag: 43 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  C...............
Jul  2 14:25:27 ezs-cocka kernel: [5191044.437278] Linux-Mag: 5c f2 6c 99 ff ff ff ff 00 4c 1f a5 03 c2 9f 81  \.l.....L.......
quade@ezs-cocka:~/linux-magazin$
```

4 Mit diesen Kommandos generieren Sie das Testmodul, laden es und sehen sich das Ergebnis an.

Fazit

Bei genauerer Betrachtung entpuppt sich die auf den ersten Blick eher schlicht aussehende Funktion `printk()` als ein sehr mächtiges, trickreiches Werkzeug, das seine Abstammung vom Urahn `printf()` keineswegs leugnet.

Das um die Funktion herum gebaute Kernel-Log-System gibt Admins alle Fäden in die Hand, um Nachrichten aus dem Linux-Inneren ressourcenschonend so zu filtern, dass keine wichtigen Nachrichten verloren gehen. Für alles darüber Hinausgehende bringt das altbewährte Syslog weitere Möglichkeiten mit. Das sind doch gute Nachrichten! (*jlu*)