

4 Kernelmodule

In dieser Aufgabe wird ein Gerätetreiber entwickelt, der gemäß der Cäsar Verschiebung verschlüsselt. Eine Verschlüsselung bildet die Buchstabenfolge eines Klartextes in die Buchstabenfolge des verschlüsselten Textes ab. Die Cäsar Verschiebung verwendet für beide Texte das selbe Alphabet. Bei diesem Verfahren erhält man den verschlüsselten Text, indem die Zeichen eines geordneten Alphabets um eine bestimmte Anzahl zyklisch nach rechts verschoben werden. In dieser Aufgabe wird folgendes geordnetes Alphabet verwendet:

ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz

Beachten Sie, **dass auch das Leerzeichen in dem Alphabet zwischen 'Z' und 'a' enthalten ist**. Gemäß Cäsar verschiebt die Verschlüsselung standardmäßig um drei Zeichen. Dies kann über einen entsprechenden Modulparameter angepasst werden (s.u.). Zeichen, die nicht im obigen Alphabet enthalten sind (wie z.B. 'ä', '?' oder 'ß'), werden unverändert bei der Verschlüsselung übernommen. Damit wir der Text

Gaius Julius!

wie folgt verschlüsselt:

JdlxvcMxolxv!

Schreiben Sie einen Gerätetreiber für zwei Geräte mit den Device-Nodes `/dev/trans0` und `/dev/trans1`. Das Device `/dev/trans0` soll als Device-Node mit der Minor-Nummer 0 angelegt werden, und verschlüsseln gemäß der Cäsar Verschiebung. Das Device `/dev/trans1` erhält die Minor-Nummer 1 und entschlüsselt einen Text.

Jeder Device-Node hat einen eigenen Puffer, der standardmäßig 40 Zeichen groß ist. Die Puffergröße ist über einen Modulparameter anpassbar (s.u.). Wird ein Text auf den Device-Node `/dev/trans0` geschrieben, so verschlüsselt das Device den Text und legt ihn in seinem Puffer gemäß FIFO ab. Ein lesender Zugriff auf `/dev/trans0` gibt den verschlüsselten Text aus. Wird ein Text auf den Device-Node `/dev/trans1` geschrieben, so entschlüsselt das Device den Text und legt ihn in seinem Puffer gemäß FIFO ab. Ein lesender Zugriff auf `/dev/trans1` gibt den entschlüsselten Text aus.

- Die beiden Geräte sollen von einem gemeinsamen Treibermodul gesteuert werden, die Funktion wird durch die *minor device number* festgelegt.
- Minor number 0 bedeutet *Verschlüsselung*.
- Minor number 1 bedeutet *Entschlüsselung*.
- Die *major device number* soll dynamisch vom Kernel vergeben werden.
- Der Treiber muss beim Laden des Moduls Pufferspeicher allokieren, der beim Entfernen des Moduls wieder freigegeben wird.

- Es darf jeweils nur ein Prozess ein Gerät zum Lesen und zum Schreiben geöffnet sein. Weitere Versuche, das Gerät zu öffnen, werden mit `–EBUSY` abgewiesen. Somit dürfen ein Gerät zum Lesen und ein Gerät zum Schreiben zeitgleich geöffnet sein. Damit muss paralleles Lesen und Schreiben auf ein Device unterstützt werden.
- Ein lesender Zugriff blockiert, wenn der entsprechende Puffer leer ist. Ein Schreibzugriff blockiert, wenn der entsprechende Puffer voll ist. Die Lese- und Schreibzeiger werden zyklisch durch den Puffer bewegt.
- Die Puffergröße `translate_bufsize` und die Anzahl der Buchstaben, um die bei der Verschlüsselung verschoben wird, `translate_shift` werden über Modulparameter festgelegt. Beim Laden ohne Parameter werden folgende Standardwerte genommen:
`translate_bufsize` = 40
`translate_shift` = 3

Hinweise

- Alles was Sie zur Bearbeitung der Aufgabe wissen müssen steht in den Kapiteln 1 – 5 sowie im Kapitel 6, Abschnitt „Blocking I/O“ des Buches „Linux Device Drivers“ von Jonathan Corbet, Alessandro Rubini, und Greg Kroah-Hartman. Die URL des ersten Kapitels ist <http://oreilly.com/catalog/linuxdrive3/book/ch01.pdf>. Aus dem Debug Kapitel benötigen Sie nur die Informationen zum Debuggen mit Text Ausgaben. Weiterhin wird in dieser Aufgabe nur ein Mutex zur Synchronisation benötigt.
- Unter <http://derekmolloy.ie/writing-a-linux-kernel-module-part-1-introduction> finden Sie Informationen, die einige Studierende im letzten Semester nutzten. Vermutlich geht diese Quelle nicht auf `copy_from_user` bzw. `copy_to_user` ein. Diesen Mechanismus sollten Sie auf jeden Fall verwenden, ansonsten tritt bei einem PageFault ein Fehler auf. Somit tritt dieser Fehler nur sporadisch auf und ist somit sehr unangenehm.
- Zugehörige Beispielprogramme finden Sie unter <http://examples.oreilly.com/linuxdrive3/examples.tar.gz>. Eine von Herrn Fohl modifizierte Version des Programms `scull` finden Sie in EMIL. Arbeiten Sie die Datei `pipe.c` durch. Dort finden Sie alles, was Sie für die Aufgabe brauchen.
- Gehen Sie großzügig mit Debug-Meldungen per `PDEBUG` um.
- Erproben Sie die Verschlüsselung zunächst in einem Userspace-Programm, das Debuggen ist da viel einfacher.
- Das Headerfile für die *Kernel*-Stringfunktionen heißt `linux / string .h`

Aufgabe

- Schreiben Sie ein Kernelmodul, das die geforderte Funktionalität realisiert.
- Das Modul *muss* aus *einer* C-Datei (und natürlich einer Header-Datei) bestehen.

- Das Modul *muss* `translate` heißen, das Kernel-Objectfile *muss* `translate.ko` heißen.
- Das Modul *muss* seine Major-Nummer dynamisch vom Kernel erhalten.

Mit anderen Worten: Sie müssen die benötigte Funktionalität aus den Quellcodefiles `main.c` und `pipe.c` zusammenklauen und in eine Datei stopfen sowie alle Erinnerungen an `scull` und `pipe` beseitigen. Außerdem müssen Sie das Makefile anpassen.

Installation

Zur Installation des Moduls schreiben Sie ein Skript, das folgendes tut:

- Alte Device-Nodes `/dev/trans?` entfernen.
- Altes Kernel-Modul entfernen.
- Neues Kernel-Modul laden.
- Major-Devicenummer aus `/proc/devices` mit `grep` und `cut` ermitteln.
- Neue Device-Nodes `/dev/trans0` und `/dev/trans1` mit der korrekten Major-Nummer erstellen.

Test / Abnahme

Führen Sie zumindest folgende Tests durch:

- Automatische Installation von Kernelmodul und Device-Nodes.
- Schreiben mit `echo` und Eingabeumleitung auf das Device, dann lesen mit `cat`.
- Beobachten der Debug-Meldungen im Kernel-Log.
- Erst Lesen mit `cat` starten, danach schreiben.
- Prüfen, ob der gleichzeitige Zugriff verhindert wird.
- Prüfen, ob die Kodierung und Dekodierung korrekt funktioniert.
- Laden des Moduls mit Parametern.