Linux-Kernel-Programmierung

Übungsaufgaben

Jürgen Quade

Linux-Kernel-Programmierung: Übungsaufgaben

von Jürgen Quade

Versionsgeschichte

Version $Revision: 1.5 $ Do 9. Jul 21:25:13 CEST 2009 Geändert durch: $Author: quade $

Version 1.0 9. Juli 2009
First draft.

Inhaltsverzeichnis

Kapitel 1. Übungen Teil 1 - Basics

1.1. Kernel-Module

1.1.1. Ziele

- Sie lernen den Umgang mit Modulen.
- Sie lernen den Aufbau einfacher Module kennen.

1.1.2. Vorbereitung

Debug-Ausgaben

Öffnen Sie eine zweite Konsole. Werden Sie Super-User und visualisieren Sie die Systemmeldungen durch folgende Eingabe:

```
(root) # tail -f /var/log/messages
```

1.1.3. Durchführung

1. Starten Sie einen Editor und geben Sie das vorgestellte Codegerüst ein. Speichern Sie die editierte Datei unter dem Namen `mod1.c`.
2. Geben Sie im Editor das vorgestellte Makefile ein. Speichern Sie die Eingaben unter dem Dateinamen `Makefile` ab. Beachten Sie die Groß- und Kleinschreibung und die Einrückung mit Tabulatoren.
3. Starten Sie den Generierungsprozess durch Eingabe von **make**.
4. Testen Sie Ihr Modul, indem Sie es laden (**insmod**), anzeigen lassen (**lsmod**) und wieder entfernen (**rmmod**). Beobachten Sie die Ausgaben im Syslog. HINWEIS: Beim Laden des Moduls wird der komplette Modulname angegeben (`mod1.ko`). Beim Entladen dagegen findet nur der Basisname Verwendung (`mod1`).

1.2. Modul-Template

1.2.1. Ziele

- Sie erstellen ein Modul-Template.
- Sie erlangen Verständnis für die Linux-Code-Besonderheiten.

1.2.2. Durchführung

1. Geben Sie das vorgestellte Code-Template ein und legen Sie dieses unter dem Dateinamen `template.c` ab.
2. Modifizieren Sie das Makefile. Ändern Sie den Namen für die zu compilierende Datei von `mod1` in `template`.
3. Starten Sie den Generierungsprozess durch Eingabe von **make**.
4. Testen Sie Ihr Modul, indem Sie es laden (**insmod**), anzeigen lassen (**lsmod**) und wieder entfernen (**rmmod**). Beobachten Sie die Ausgaben im Syslog.

1.3. Vom Modul zum Treiber

1.3.1. Ziele

- Sie lernen den Unterschied zwischen einem Modul und einem Treiber.
- Sie lernen die Besonderheiten des Modul-Codes kennen.

1.3.2. Vorbereitung

Quelldatei erstellen

Kopieren Sie das in der vorhergehenden Übung erstellte Template in die Datei `treiber.c`.

Makefile anpassen

Modifizieren Sie das Makefile. Ändern Sie den Namen für die zu compilierende Datei in `treiber`.

1.3.3. Durchführung

1. Erweitern Sie den Code in `treiber.c` um die Funktion, die das Modul beim Kernel als Treiber anmeldet (`register_chrdev()`). HINWEIS: Die notwendigen Prototypen sind in der Headerdatei `<linux/fs.h>` deklariert.
2. Erweitern Sie den Code in `treiber.c` um die Funktion, die das Modul beim Kernel als Treiber wieder abmeldet (`unregister_chrdev()`).
3. Starten Sie den Generierungsprozess durch Eingabe von **make**.
4. Laden Sie das generierte Modul (**(root)# insmod treiber.ko**). Überprüfen Sie, ob sich Ihr Modul erfolgreich als Treiber angemeldet hat, indem Sie sich die geladenen Treiber durch Aufruf des Kommandos **(root)# cat /proc/devices** ansehen. Beobachten Sie außerdem die Ausgaben im Syslog.

1.4. Open und Release im Treiber

1.4.1. Ziele

- Sie sollen die Bedeutung und den Aufbau der Open- und Close-Funktionen kennen lernen.
- Sie lernen die Open- und Close-Funktionen zu kodieren.
- Sie lernen den Zugriff auf Treiber mit Hilfe selbst geschriebener Applikationen.

1.4.2. Vorbereitung

Quelldatei erstellen

Erstellen Sie auf Basis des Templates eine Datei namens `openclose.c`.

Makefile anpassen

Modifizieren Sie das Makefile. Ändern Sie den Namen für die zu compilierende Datei in `openclose`.

Syslog ausgeben

Halten Sie weiterhin das Fenster mit den Syslog-Ausgaben offen.

1.4.3. Durchführung

1. Erweitern Sie das Codefragment um eine Open-Funktion, die nur eine `printk`-Ausgabe macht.
2. Erweitern Sie den Code um eine Release-Funktion. Auch diese soll nur eine `printk` Ausgabe machen.
3. Starten Sie den Generierungsprozess durch Eingabe von
   ```
   (root)# make
   ```
4. Legen Sie eine Gerätedatei an:
   ```
   (root)# mknod geraetedatei c 240 0
   ```
5. Greifen Sie auf die Gerätedatei (und damit auf den Treiber) mit dem Systemprogramm **cat** zu.
6. Erweitern Sie Ihren Treiber um einen Zugriffsschutz: Zu einem Zeitpunkt darf immer nur ein Rechenprozess zugreifen. HINWEIS: Für die Realisierung des Zugriffsschutzes verwenden Sie eine globale Variable (entweder als Flag oder als Zähler). Im Vorgriff auf eine spätere Lerneinheit definieren Sie die Variable vom Typ `atomic_t`. Sie können folgende Prototypen verwenden:
   ```
   static atomic_t access_count = ATOMIC_INIT(-1);

   atomic_inc_and_test( atomic_t *v); // v++; if( v==0 ) return 1
   atomic_dec( atomic_t *v );         // v--;
   ```
7. Starten Sie den Generierungsprozess durch Eingabe von **make**.
8. Laden Sie Ihren Treiber und testen Sie diesen mit Hilfe des Systemkommandos **cat**.
9. Erstellen Sie eine eigene Applikation (mit Hilfe der Systemcalls **open**, **close**, **read** und **write**).
10. Testen Sie den Treiber mit Hilfe der selbst erstellten Applikation.

1.5. Lesefunktion

1.5.1. Ziele

- Hier lernen Sie im Treiber die Funktion für den lesenden Zugriff zu realisieren.
- Sie lernen, wie Daten zwischen Kernel- und User-Space ausgetauscht werden.

1.5.2. Vorbereitung

Quelldatei erstellen

: Erstellen Sie auf Basis des Templates eine Datei namens `hello.c`.

Makefile anpassen

: Modifizieren Sie das Makefile. Ändern Sie den Namen für die zu compilierende Datei in `hello`.

Syslog ausgeben

: Halten Sie weiterhin das Fenster mit den Syslog-Ausgaben offen.

1.5.3. Durchführung

1. Erweitern Sie das Codefragment um eine Lese-Funktion, die beim lesenden Zugriff einer Applikation den String `Hello World` zurückgibt.
2. Starten Sie den Generierungsprozess durch Eingabe von **make**.
3. Testen Sie den Treiber mit dem Systemprogramm **cat**. Ihr Treiber liefert ständig »hello world« zurück. Warum?
4. Testen Sie den Treiber mit Hilfe einer selbst erstellten Applikation.

1.6. Virtuelles Gerät »/dev/zero«

1.6.1. Ziele

- Hier lernen Sie, wie das virtuelle Gerät »/dev/zero« implementiert wird.
- In der Zusatzaufgabe lernen Sie die Auswertung der Minornummer.

1.6.2. Vorbereitung

Quelldatei erstellen

Erstellen Sie auf Basis des Templates eine Datei namens `zero.c`.

Makefile anpassen

Modifizieren Sie das Makefile. Ändern Sie den Namen für die zu compilierende Datei in `zero`.

Gerätedatei

Stellen Sie sicher, dass eine Gerätedatei mit dem Namen `geraetedatei` und der Major-/Minornummer 240, 0 existiert.

Erstellen Sie für die Zusatzaufgabe außerdem eine Gerätedatei mit dem Namen `myzero` und der Major-/Minornummer 240, 1.

1.6.3. Durchführung

1. Erweitern Sie das Codefragment um eine Lese-Funktion, die beim lesenden Zugriff einer Applikation immer ein »0« zurückgibt.
2. Testen Sie den Treiber mit den Systemprogrammen **cat** und **hexdump**.
3. Zusatzaufgabe: Die Lesefunktion soll nur aktivierbar sein, wenn die Minornummer »1« beträgt. Wird der Treiber mit einer Majornummer »0« aufgerufen, gibt er „hello world“ zurück.

```
minornumber = iminor(instanz->f_dentry->d_inode);
```

1.7. Schreibfunktion und »/dev/null«

1.7.1. Ziele

- Hier lernen Sie, wie der schreibende Zugriff auf den Treiber realisiert wird.
- Hier lernen Sie, wie das virtuelle Gerät »/dev/null« realisiert ist.

1.7.2. Vorbereitung

Quelldatei erstellen

Erstellen Sie auf Basis des Templates eine Datei namens `null.c`.

Makefile anpassen

Modifizieren Sie das Makefile. Ändern Sie den Namen für die zu compilierende Datei in `null`.

1.7.3. Durchführung

1. Erweitern Sie das Codefragment um eine Schreib-Funktion (`driver_write()`), die alle übergebenen (zu schreibenden) Daten »verschluckt«.
2. Testen Sie den Treiber mit dem Systemprogramm **echo**.

1.8. Zugriffsmodi

1.8.1. Ziele

- Sie lernen die generelle Struktur einer applikationsgetriggerten Funktion kennen.
- Hier lernen Sie, die Realisierung der Zugriffsmodi »blockierend« und »nicht blockierend« im Treiber kennen.
- Sie lernen eine erste Behandlung von Signals im Treiber kennen.

1.8.2. Vorbereitung

Quelldatei erstellen

Erstellen Sie auf Basis des Templates eine Datei namens `buf.c`.

Makefile anpassen

Modifizieren Sie das Makefile. Ändern Sie den Namen für die zu compilierende Datei in `buf`.

Syslog ausgeben

Öffnen Sie ein Fenster mit den Syslog-Ausgaben (**tail -f /var/log/messages**).

1.8.3. Durchführung

1. Erweitern Sie das Codefragment um eine Schreib-Funktion (`driver_write()`), die alle übergebenen (zu schreibenden) Daten in einen internen Buffer ablegt.
2. Erweitern Sie das Codefragment um eine Lese-Funktion, die die im internen Buffer befindlichen Daten der aufrufenden Applikation übergibt. Um die Aufgabe möglichst einfach zu halten, sollen immer die zuletzt geschriebenen Daten (die jüngsten Daten also) zurückgegeben werden. Außerdem soll auf den an sich notwendigen Schutz des Zugriffes auf den gemeinsamen Speicher verzichtet werden.

   Übergebene Daten sollen aus dem Buffer gelöscht werden. Sind im Buffer keine Daten vorhanden, soll die Applikation, die `driver_read` aufgerufen hat, schlafen gelegt werden.

```
static wait_queue_head_t wq;
...
    if( wait_event_interruptible(wq,(pos>0)) )
        return -ERESTART;
...
```

```
    wake_up( &wq );
...
    init_waitqueue_head( &wq );
```

3. Starten Sie den Generierungsprozess des Treibers durch Eingabe von **make**. Laden Sie den Treiber.
4. Testen Sie den Treiber indem Sie folgendermassen vorgehen:
   a. Lesen Sie von dem durch den Treiber bediente Gerät (**cat geraetedatei**). Da der Buffer leer ist, muss die lesende Applikation vom Treiber schlafen gelegt werden.
   b. Schreiben Sie Daten auf das durch den Treiber bediente Gerät (z.B. durch »**echo hallo >geraetedatei**«).

      Jetzt muss durch den ersten Prozess die Ausgabe des Leseaufrufes erscheinen (der lesende Prozess ist durch den Kernel aufgeweckt worden). Danach legt sich der lesende Prozess jedoch gleich wieder schlafen.

1.9. Instanzenspezifische Parameter

1.9.1. Ziele

- Sie lernen die Bedeutung von instanzenspezifischen Parametern kennen.
- Hier lernen Sie, wie Sie instanzenspezifische Parameter in Ihrem Treiber ablegen und verwenden.

1.9.2. Vorbereitung

Quelldatei erstellen

Kopieren Sie den Quellcode des Treibers »hello.c« in eine Datei namens `hello2.c`.

Makefile anpassen

Modifizieren Sie das Makefile. Ändern Sie den Namen für die zu compilierende Datei in `hello2`.

Syslog ausgeben

Öffnen Sie ein Fenster mit den Syslog-Ausgaben (**tail -f /var/log/messages**).

1.9.3. Durchführung

1. Erweitern Sie den »Hello-World-Treiber« derartig, dass jede aufrufende Treiberinstanz zwischen `open()` und `close()` genau einmal den String „Hello World“ liest.

   Erstellen Sie hierzu eine Datenstruktur, in der die bereits ausgegebenen Zeichen abgespeichert werden können. Wenn der komplette String ausgelesen wurde, gibt `driver_read()` `0` (End Of File) zurück.

   Reservieren Sie den notwendigen Speicher durch Aufruf der Funktion `kmalloc(sizeof(struct idata), GFP_USER)` in der Funktion `driver_open()`.

Vergessen Sie nicht, innerhalb der Funktion `driver_close()` den reservierten Speicher wieder freizugeben.

2. Testen Sie Ihren Treiber, in dem Sie selbst eine Applikation schreiben, die ein `open()` aufruft und dann nach dem Lesen noch 10 Sekunden schläft. In diesen 10 Sekunden starten Sie auf einer anderen Konsole ein **cat geraetedatei**. Beide Programme müssen genau einmal „Hello World“ lesen.

1.10. Tasklets

1.10.1. Ziele

- Sie lernen die Realisierung von Tasklets.
- Sie lernen, wie Tasklets aktiviert werden.
- Sie lernen die Probleme bei der Verwendung von Tasklets (vorzeitiges Entladen des Tasklet-Codes).

1.10.2. Vorbereitung

Quelldatei erstellen

Erstellen Sie auf Basis des Templates eine Datei namens `tasklet.c`.

Makefile anpassen

Modifizieren Sie das Makefile. Ändern Sie den Namen für die zu compilierende Datei in `tasklet`.

Syslog ausgeben

Öffnen Sie ein Fenster mit den Syslog-Ausgaben (**tail -f /var/log/messages**).

1.10.3. Durchführung

1. Implementieren Sie ein Tasklet, welches nur per `printk()` eine Ausgabe in den Syslog macht. HINWEIS: Vergessen Sie nicht den Code für das Deaktivieren des Tasklets beim Entladen des Moduls (`tasklet_kill`).
2. Starten Sie den Generierungsprozess des Treibers durch Eingabe von **make**.
3. Testen Sie den Code indem Sie den Treiber laden und die Ausgabe im Syslog beobachten.

1.11. Timer

1.11.1. Ziele

- Hier lernen Sie das Aufsetzen von Timer-Funktionen im Kernel.
- Sie lernen die Probleme bei der Verwendung von Timer (vorzeitiges Entladen des Timer-Codes).
- Sie lernen den Umgang mit Jiffies.

1.11.2. Vorbereitung

Quelldatei erstellen

: Erstellen Sie auf Basis des Templates eine Datei namens `timer.c`.

Makefile anpassen

: Modifizieren Sie das Makefile. Ändern Sie den Namen für die zu compilierende Datei in `timer`.

Syslog ausgeben

: Öffnen Sie ein Fenster mit den Syslog-Ausgaben (**tail -f /var/log/messages**).

1.11.3. Durchführung

1. Implementieren Sie einen Timer, welcher alle zwei Sekunden aufgerufen wird. Mit jedem Aufruf soll der Timer die Zeit (in Jiffies) seit dem letzten Aufruf im Syslog ausgeben. Berechnen Sie ebenfalls die bisherige Minimal- und die Maximalzeit und geben Sie diese ebenfalls aus.
2. Starten Sie den Generierungsprozess des Treibers durch Eingabe von **make**.
3. Testen Sie den Code indem Sie den Treiber laden und die Ausgabe im Syslog beobachten.
4. Zusatzaufgaben: Geben Sie die Differenzzeit zum letzten Aufruf und deren Maximal bzw. Minimalwert über den Taktzyklenzähler aus (Funktionen `rdtscl(unsigned long low)` und `unsigned int cpufreq_get(unsigned int cpu)`).

1.12. Kernel-Threads

1.12.1. Ziele

- Sie lernen den prinzipiellen Aufbau eines Kernel-Threads kennen.
- Sie lernen einen Kernel-Thread zu implementieren.
- Sie lernen, wie ein Kernel-Thread wieder entfernt wird.

1.12.2. Vorbereitung

Quelldatei erstellen

Erstellen Sie auf Basis des Templates eine Datei namens `kthread.c`.

Makefile anpassen

Modifizieren Sie das Makefile. Ändern Sie den Namen für die zu compilierende Datei in `kthread`.

Syslog ausgeben

Öffnen Sie ein Fenster mit den Syslog-Ausgaben (**tail -f /var/log/messages**).

1.12.3. Durchführung

1. Implementieren Sie einen Kernel-Thread, der sich jeweils für zwei Sekunden schlafen legt und – jedesmal, wenn er geweckt wird – eine Ausgabe ins Syslog macht. Der Kernel-Thread soll sich beenden, sobald ihm ein Signal geschickt wird. Zum Ende ruft er die Funktion `complete_and_exit( &on_exit, 0 )` auf.

   Stellen Sie sicher, dass beim Entladen des zugehörigen Moduls ein solches Signal an den Thread geschickt wird (Funktion `kill_proc( thread_id, SIGTERM, 1 );` und warten Sie auf das Ende des Kernel-Threads (`wait_for_completion( &on_exit )`.
2. Starten Sie den Generierungsprozess des Treibers durch Eingabe von **make** und laden Sie das generierte Modul.
3. Beobachten Sie die Ausgaben im Syslog.
4. Identifizieren Sie die PID des Kernel-Threads (z.B. über das Kommando **ps -awxu**).
5. Schicken Sie Ihrem Kernel-Thread über die Konsole ein Signal (**kill <PID>**). Beobachten Sie die Ausgaben im Syslog und die Taskliste.
6. Wird der Kernel-Thread beim Entladen des Moduls ordnungsgemäss entfernt?

1.13. Workqueues

1.13.1. Ziele

- In dieser Übung lernen Sie den Umgang mit Workqueues kennen.

1.13.2. Vorbereitung

Quelldatei erstellen

Erstellen Sie auf Basis des Templates eine Datei namens `wq.c`.

Makefile anpassen

Modifizieren Sie das Makefile. Ändern Sie den Namen für die zu compilierende Datei in `wq`.

Syslog ausgeben

Öffnen Sie ein Fenster mit den Syslog-Ausgaben (**tail -f /var/log/messages**).

1.13.3. Durchführung

1. Instanzieren Sie (in der Modul-Initialisierungsfunktion) eine Workqueue mit dem Namen `meine_wq` (Funktion `create_workqueue()`). Instanzieren Sie außerdem ein Work-Objekt (`DECLARE_WORK(...)`). Schreiben Sie eine Funktion, die nur eine `printk`-Ausgabe macht. Lassen Sie diese Funktion im Kontext der Workqueue nach 4 Sekunden aufrufen.

   Stellen Sie sicher, dass beim Entladen des Moduls Workobjekt und Workqueue ordnungsgemäss aus dem System entfernt werden.

   ```
   cancel_delayed_work( &work_obj );
   flush_workqueue( wq );
   destroy_workqueue( wq );
   ```

2. Starten Sie den Generierungsprozess des Treibers durch Eingabe von **make** und laden Sie das generierte Modul.
3. Unter welchem Namen findet sich die Workqueue in der Taskliste wieder?
4. Beobachten Sie die Ausgaben im Syslog. Laden Sie das Modul und entladen Sie es direkt wieder. Bleibt das System stabil?

1.14. Der Schutz kritischer Abschnitte

1.14.1. Ziele

- In dieser Übung lernen Sie weitergehende Techniken zum Schutz kritischer Abschnitte kennen.
- Sie lernen den Einsatz den Umgang mit Kernel-Mutexen.

1.14.2. Vorbereitung

Quelldatei erstellen

Kopieren Sie den Quellcode des Treibers »openclose.c« in eine Datei namens `mutex.c`.

Makefile anpassen

Modifizieren Sie das Makefile. Ändern Sie den Namen für die zu compilierende Datei in `mutex`.

Syslog ausgeben

Öffnen Sie ein Fenster mit den Syslog-Ausgaben (**tail -f /var/log/messages**).

Gerätedatei

Stellen Sie sicher, dass eine Gerätedatei mit der Majornummer 240 existiert.

1.14.3. Durchführung

1. Erstellen Sie einen Gerätetreiber mit einer `driver_open`-Funktion. Legen Sie die zugehörige Instanz (Rechenprozess) innerhalb der `driver_open`-Funktion für 3 Sekunden schlafen. Stellen Sie mit Hilfe eines Mutex sicher, dass die Funktion `driver_open` zu einem Zeitpunkt nur von einer Instanz geöffnet werden kann. Damit ein erfolgloser Versuch, den kritischen Abschnitt zu betreten, protokolliert werden kann, verwenden Sie die Funktion `mutex_trylock()`. Ist der kritische Abschnitt belegt, soll in einer Schleife mit 200 ms Sekunde Wartepause solange versucht werden den kritischen Abschnitt zu betreten, bis dieser frei ist.
2. Starten Sie den Generierungsprozess des Treibers durch Eingabe von **make** und laden Sie das generierte Modul.
3. Testen Sie Ihren Treiber, in dem Sie mehrfach per **cat** die zugehörige Gerätedatei öffnen. Beobachen Sie die Ausgaben im Syslog.

1.15. Sys-Filesystem

1.15.1. Ziele

- Sie lernen den Aufbau des Gerätemodells und die Repräsentierung über das Sys-Filesystem kennen.
- Sie lernen Informationen zwischen User- und Kernelspace über das Sys-Filesystem auszutauschen.

1.15.2. Vorbereitung

Quelldatei erstellen

Kopieren Sie den Treiber aus Übung »Zugriffsmodi« (Datei `buf.c`) in eine Datei namens `sys.c`.

Makefile anpassen

Modifizieren Sie das Makefile. Ändern Sie den Namen für die zu compilierende Datei in `sys`.

Gerätedatei

Stellen Sie sicher, dass eine Gerätedatei mit der Majornummer 240 existiert.

1.15.3. Durchführung

1. Erweitern Sie den Treiber `sys.c`, so dass sich dieser als ein Treiber für die Platform im Sys-Filesystem unter dem Verzeichnis `/sys/bus/platform/stat` anmeldet. Dazu müssen Sie ein Objekt vom Typ

`struct device_driver` instanzieren und dem Kernel durch Aufruf der Funktion `driver_register(...)` übergeben.

2. Erweitern Sie den Treiber `sys.c`, so dass eine Statistik über die Anzahl der Aufrufe von `driver_open`, `driver_close`, `driver_write` und `driver_read` geführt wird. Achten Sie darauf, dass es beim Zugriff auf die Statistik-Variablen eine Race-Condition geben kann! Verwenden Sie daher Variablen vom Typ `atomic_t`.
3. Implementieren Sie eine Attributdatei `access`, die die gesammelte Statistik (beim lesenden Zugriff) ausgibt. Hierzu müssen Sie als erstes die Lesefunktion realisieren, danach das Dateiobjekt erstellen und dieses schließlich beim Kernel anmelden:

```
...
static ssize_t stat_access( struct device_driver *driver, char *buffer )
{
    ...
}

static DRIVER_ATTR( access, S_IRUGO, stat_access, NULL );
...
    driver_create_file( &buf_driver, &driver_attr_access );
...
```

4. Implementieren Sie eine Attributdatei `reset`, die die Zähler der Statistik (beim schreibenden Zugriff) wieder zu Null setzt.

1.16. Udev-Connection

1.16.1. Ziele

- Sie lernen, wie ein Treiber das automatische Erzeugen einer Gerätedatei durch Udev unterstützt.
- Sie lernen, wie im Gerätemodell eigene Klassen angelegt werden.

1.16.2. Vorbereitung

Quelldatei erstellen

Kopieren Sie den Treiber aus Übung »Zugriffsmodi« (Datei `buf.c`) in eine Datei namens `udev.c`.

Makefile anpassen

Modifizieren Sie das Makefile. Ändern Sie den Namen für die zu compilierende Datei in `udev`.

1.16.3. Durchführung

1. Verändern Sie den Treiber `buf.c`, dass dieser sich vom Kernel dynamisch eine Majornummer zuteilen lässt (Aufruf der Funktion `register_chrdev()` mit Majornummer `0`, der Rückgabewert ist dann die zugeteilte Majornummer).

2. Erweitern Sie den Treiber `buf.c` um eine Geräteklasse mit Namen »company«. Übersetzen und laden Sie den Treiber. Überprüfen Sie, dass die neue Geräteklasse im Sys-Filesystem erscheint.
3. Melden Sie den Treiber beim Gerätemodell unter der neuen Klasse »company_class« und dem Namen »lulu« an. Vergessen Sie nicht, auch das Abmelden zu implementieren.
4. Testen Sie den Treiber, indem Sie über die vom udev erzeugte Gerätedatei zugreifen.

1.17. Proc-Filesystem

1.17.1. Ziele

- Sie lernen Informationen zwischen User- und Kernelspace über das Proc-Filesystem auszutauschen.

1.17.2. Vorbereitung

Quelldatei erstellen

Kopieren Sie den Treiber aus Übung »Sys-Filesystem« (Datei `sys.c`) in eine Datei namens `sysproc.c`.

Makefile anpassen

Modifizieren Sie das Makefile. Ändern Sie den Namen für die zu compilierende Datei in `sysproc`.

1.17.3. Durchführung

1. Erweitern Sie den Treiber `sysproc.c`, so dass dieser Treiber für die Platform im Sys-Filesystem unter dem die erfasste Statistik auch über die Proc-Datei Verzeichnis `/proc/access-stat/access` ausgibt. Hierzu müssen Sie als erstes die Proc-Lesefunktion realisieren und diese hernach dem Kernel übergeben.
2. Zusatzaufgabe: Implementieren Sie auch die Reset-Funktionalität in Form einer Proc-Datei. Die folgenden Definitionen helfen Ihnen:

```
static int proc_write( struct file *instanz, const char *ubuf,
unsigned long count, void *data )
{
...
    proc_reset = create_proc_entry( "reset", S_IWUGO, proc_dir );
    if( proc_reset ) {
        proc_reset->write_proc = proc_write;
    }
...
    if( proc_reset ) remove_proc_entry( "reset", proc_dir );
...
```