Studiengang BIS15 VZ Betriebssysteme und Systemprogrammierung

Übungsblatt 6 Thema: Kernelprogrammierung – Kennenlernen der Schnittstelle WS 2016/17

Der Treiber dieser Übung muss mehrere Geräte unterstützen.

Achten Sie besonders darauf, dass keine Race Conditions auftreten können, also synchronisieren sie, wo nötig!!!

Übungsaufgabe:
Implementieren Sie einen einfachen Charakter Device Treiber, der **mehrere** virtuelle Geräte im Arbeitsspeicher unterstützt und der folgendes leistet:

Unterstützt werden die Systemcalls open, close, read, write und ioctl.

Tipp: Benutzen Sie das Linux Device Drivers Buch (Pdf am eCampus) als Nachschlagewerk. Speziell die Kapiteln 2, 3 und 6.

Ein virtuelles Gerät hat eine Speicherkapazität von 1024 Byte, die als Puffer dieser Größe beim Gerät organisiert werden. Zu Beginn ist das Gerät leer (hat also 0 gültige Bytes Inhalt) und enthält logisch gesehen in der Folge je nach ***write*** und ***ioctl*** Systemcalls immer zwischen 0 und 1024 gültige Bytes an Inhalt.
Der tatsächliche Speicherplatz soll erst allokiert werden, wenn der 1. Open Systemcall zum Schreiben erfolgt! Der Speicherplatz bleibt aber dann für das Gerät reserviert, bis der Treiber entladen wird! (Wird das Gerät nie geöffnet werden, belegt es für den Puffer auch keinen Platz.)

Open:
Beim Öffnen zum Schreiben wird im (virtuellen) Gerät ein Puffer mit 1024 Byte Größe allokiert, falls er noch nicht existiert; und im Falle dieses „erstmaligen Öffnens“ des Geräts ist zu vermerken, dass der (logische) Inhalt des Puffers leer ist. Ein Gerät darf gleichzeitig immer nur von einem Prozess zum Schreiben geöffnet sein! Versucht ein weiterer Prozess das gleiche Gerät zum Schreiben zu öffnen, das schon zum Schreiben offen ist, bekommt er über errno ein „Device BUSY“ gemeldet.

Das Öffnen zum Lesen wird nur dann fehlerfrei durchgeführt, wenn der Daten-Puffer dieses Geräts existiert (also zumindest 1 Mal ein Prozess das Gerät schon zum Schreiben geöffnet hatte. Es können aber auch mehrere Open zum Lesen gemacht werden. Jedes Open „landet“ dabei an der „aktuellen“ Position 0 des Puffers und ist in der Folge in der Lage den enthaltenen Inhalt auszulesen.

Write:
Das Write ist so zu implementieren, dass die zu schreibenden Bytes im Puffer an der aktuellen Position geschrieben bzw. angehängt werden (falls die akt Position am Ende steht). Achten Sie aber darauf, dass das auch im Puffer noch Platz hat!! Sind nicht genügend Bytes im Buffer verfügbar, sollen die letzten Bytes aufgebraucht werden und entsprechend die Anzahl der wirklich geschriebenen Bytes zurück gegeben werden. Falls kein einziges Byte mehr im Buffer verfügbar ist, können sie 0 oder einen passenden Error-Code zurückgeben (zB: „Device Busy“). Achten Sie darauf, dass an der aktuellen Position beim nächsten write weitergeschrieben wird.
Ein neues Öffnen des Devices und anschließendes Schreiben, überschreibt den bestehenden Inhalt, aber löscht zuvor nicht den alten Inhalt. Also wenn „asdf“ und „1234“ geschrieben wird, steht „asdf1234“ im Buffer. Wird das Device geschlossen und neu zum Schreiben geöffnet und „xy“

geschrieben, steht „xydf1234“ im Buffer. Ist die aktuelle Dateiposition außerhalb des aktuellen Inhalts (weil inzwischen mit ioctl der Inhalt gelöscht wurde), soll die aktuelle Dateiposition auf das aktuelle Ende des Inhalts gesetzt werden.

Read:
Gelesen werden so viele Bytes, wie der Aufrufer wünscht bzw. halt noch da sind.
Achten Sie darauf, dass an der aktuellen Position weitergelesen wird. Ein EOF ist dann zu erkennen, wenn bereits alles was in den Puffer geschrieben wurde auch gelesen wurde und ein weiterer Leseversuch gemacht wird. Ist die aktuelle Dateiposition außerhalb des aktuellen Inhalts (weil inzwischen mit ioctl der Inhalt gelöscht wurde) soll die akt Leseposition unverändert bleiben und Sie können 0 oder einen passenden Fehlercode zurück geben.

Ioctl:
Mit ioctl kann

a) abgefragt werden, wie oft das Gerät gerade zum Lesen offen ist,
b) abgefragt werden, ob das Gerät gerade zum Schreiben offen ist,
c) der Inhalt des Geräts gelöscht werden. Andere Prozesse, die das Gerät gerade offen haben sollen sich wie in Write und Read beschrieben, anschließend verhalten.

Testprogramm:
Zum Testen des Treibers soll ein Testprogramm geschrieben werden, dass zum Schreiben auf das Gerät bzw. zum Lesen vom Gerät dient. Testprogramm soll auch die möglichen **ioctl**-Aufrufe beinhalten, um den „Treiber“ testen zu können.
Zum Testen können sich auch zusätzlich mit echo und cat ihr jeweiliges device testen. (echo „abc“ > /dev/mydevX etc.)

<u>Zu entwickeln und abzugeben sind:</u>

Der gut dokumentierte Kerneltreiber samt Makefile und Shellskripts zum Laden und Entladen des Moduls und das Testprogramm.