Fachprojekt „Systemsoftwaretechnik“
A3 – Treiber

3 Der eigene Linux-Treiber

In dieser Aufgabe werdet Ihr keine Fehler suchen – zumindest keine, die wir eingebaut haben. Stattdessen sollt Ihr in dieser Aufgabe selbst einen kleinen Treiber schreiben. Ladet Euch hierzu mittels `git` den Quellcode für die Aufgaben aus der Vorgabe[1] herunter, sofern Ihr dies noch nicht für den vorherigen Zettel getan habt. Achtet auf die Hinweise am Ende des Übungsblatts.

Das Grundgerüst für Euren Treiber liegt im Zweig `a3` im Verzeichnis `drivers/student`. Benutzt als Ausgangspunkt die Konfiguration `a3.config`. Die Vorgabe enthält bereits die Konfigurationsoptionen. Ihr müsst lediglich den entsprechenden Treiber aktivieren. Schaut einmal in `drivers/student/Kconfig` nach, wie die Konfigurationsoption heißt, um sie anschließend im Konfigurationsmenü suchen zu können.

Legt das Quellcodeverzeichnis ruhig in Eurer Home-Verzeichnis ab. Baut den Kern aber dann *bitte unbedingt* mit dem Parameter `O=/fs/scratch/XX/YY`. Die beiden Platzhalter entsprechen Eurem Benutzernamen und einem beliebigen Unterverzeichnis. **Wichtig:** In dem Fall *muss* die Konfigurationsdatei in dem oben genannten Verzeichnis liegen und Ihr müsst die Option bei jedem Aufruf von `make` angeben.

3.1 Anforderungen

Es ist komplett Euch überlassen, welche konkrete Funktionalität Euer Treiber erfüllt. In diesem Abschnitt findet Ihr lediglich die Anforderungen, die Euer Treiber erfüllen muss. Falls Ihr Ideen oder Hilfestellung benötigt, findet Ihr in Abschnitt 3.2 ein exemplarisches Vorgehen für einen Treiber.

- Erstellt eine Konfigurationsoption, die a) von der Option `STUDENT` abhängt und b) einen Wert (z. B. `int` oder `string`) zugewiesen bekommt. Baut diese Option in Euren Quelltext ein.
- Fügt Eurem Treiber eine Lese- und eine Schreib-Funktion hinzu, in denen Ihr Inhalt von bzw. zum Userspace kopiert.
- Baut mindestens drei konfigurierbare Debug-Ausgaben, wie Ihr sie auf Blatt A1 kennengelernt habt, ein. Definiert zusätzlich Eure eigene Version des Makros `pr_fmt` mit einem *eigenen* Text.
- Alloziert zur Laufzeit Speicher und gebt ihn natürlich wieder frei[2].

Abseits dieser Anforderungen steht es Euch vollkommen frei, weitere Dinge, wie beispielsweise weitere Funktionszeiger (`open()`, ...) oder Threads, zu verwenden. Fragt Euren Tutor gerne nach weiterer Dokumentation!

[1] https://git.cs.tu-dortmund.de/sst-ss24/linux
[2] Falls nicht, wisst Ihr ja, wie Ihr das Speicherleck findet. ☺

3.2 Beispiel

Wenn Euch eine passende Idee fehlt und/oder Ihr (noch) Respekt vor dem Linux-Kern habt, implementiert doch z. B. einen Treiber mit folgender Funktionalität:

Der Treiber liest zwei aufeinanderfolgende Zahlen durch ein Schreiben in die Gerätedatei einen *globalen* Buffer ein. Diesen alloziert Ihr zur Laufzeit, wenn dieser Buffer nicht existiert. Anschließend wird eine zur Übersetzungszeit feste mathematische Operation angewandt, z. B. immer die Summe berechnen. Beim Lesen aus der Gerätedatei wird das Ergebnis der Summe zurückgegeben. Am Ende der Lese-Operation gebt Ihr den *globalen* Buffer frei. Ihr erstellt eine `int`-Konfigurationsoption, die einen Offset repräsentiert und bei jeder Operation z. B. drauf addiert wird.

Optional erweitert Ihr die Anzahl an Zahlen von zwei auf n und legt z. B. den Punkt als Terminator einer Zahlenfolge fest. Erst bei einem Punkt wird die gewünschte Operation durchgeführt.

Allgemeine Hinweise:

- Falls der Bildschirm nicht ausreicht, um eine Fehlermeldung vollständig zu betrachten, könnt Ihr die serielle Konsole nutzen. Hierzu müsst Ihr dem Linux-Kern einen zusätzlichen Parameter übergeben[a]. Schaut Euch neben der Dokumentation auch die Hilfeausgabe des Skripts `boot.sh` an. Diese zeigt Euch, wie Ihr Parameter an den Kern übergeben könnt.
- Achtet darauf, dass Ihr das Pseuo-Dateisystem `debugfs` aktiviert und eingehangen habt. Letzteres erreicht Ihr mit dem Befehl: `mount -t debugfs none /sys/kernel/debug`. Möchtet Ihr das Verzeichnis `vmshare` bei jedem Start einhängen, ändert die Datei `/etc/fstab`:`vmshare /mnt 9p trans=virtio 0 2`
- Das `boot.sh`-Skript leitet den Netzwerkport 22 an den Port `4711` des Hosts weiter. So könnt Ihr Euch, sofern SSH installiert ist, via SSH in der `VM` anmelden. Ihr müsst dem `ssh`-Kommando nur die richtigen Parameter Port angeben – siehe `man ssh`. Verwendet als Host `localhost`. Wenn Ihr außerdem in `/etc/ssh/sshd_config` die Variable `PermitRootLogin` einkommentiert und auf `yes` setzt, könnt Ihr Euch direkt als `root` per SSH anmelden.

[a] https://www.kernel.org/doc/html/latest/admin-guide/serial-console.html