Zum Auftakt booten (oder reaktivieren) Sie die Ulix-Devel-VM und führen in der Shell den Befehl `update-ulix.sh` aus. Damit laden Sie die Dateien herunter, die Sie für das Bearbeiten der aktuellen Übungsaufgaben benötigen.

19. Tests des Festplattentreibers

In `tutorial08/` liegt jetzt eine aktualisierte Ulix-Version, welche die Funktionen `readsector_hd` und `writesector_hd` für den Festplattenzugriff implementiert. Das `Makefile` wurde so angepasst, dass die Ulix-VM beim Start zwei virtuelle Festplatten `hda.img` und `hdb.img` verwendet.

a) Füllen Sie `hda.img` mit beliebigen Inhalten, z. B. indem Sie eine andere Datei nach `hda.img` kopieren. Die Datei sollte mehrere KByte groß sein. Fügen Sie dann im Code Chunk *<kernel main: user defined tests>* Aufrufe von `readsector_hd()` ein, welche verschiedene Sektoren der ersten Festplatte einlesen. Geben Sie diese zur Kontrolle mit `printf()` auf dem Bildschirm aus und vergleichen Sie die Ausgabe mit dem Inhalt von `hda.img`. (Bei der Ausgabe eines Sektors mit `printf()` sollten Sie dafür sorgen, dass am Ende ein `\0`-Zeichen steht, damit die Ausgabe auf jeden Fall abbricht.)

Um die Sektor-Positionen der Inhalte zu bestimmen, können Sie die Image-Datei mit

```
hexdump -C hda.img | less
```

betrachten: In Hexadezimalschreibweise ist 512 `0x200`, d. h., dass neue Sektoren bei `0x200`, `0x400`, `0x600` usw. beginnen.

b) Im nächsten Schritt kopieren Sie (in Ulix) mit `readsector_hd()` und `writesector_hd()` die gesamte erste Festplatte auf die zweite Festplatte. Dazu muss `hdb.img` beim Start von Ulix bereits die richtige Dateigröße haben, denn Sie können nicht „über das Ende der Festplatte“ hinaus schreiben.

Um `hdb.img` mit *X* KByte an Null-Bytes (`\0`) zu füllen, können Sie das Kommando

```
dd if=/dev/zero of=hdb.img bs=1k count=X
```

verwenden. Vergleichen Sie nach der Kopieraktion in Ulix mit `diff hda.img hdb.img` die beiden Image-Dateien – falls das Tool einen Unterschied entdeckt, können Sie mit den folgenden Kommandos Hexdumps der Dateien erstellen und diese vergleichen:

```
hexdump -C hda.img > hda.txt
hexdump -C hdb.img > hdb.txt
diff hda.txt hdb.txt
```

(In Textdateien kann `diff` Unterschiede konkret anzeigen, bei Binärdateien kann es nur „Dateien unterscheiden sich“ antworten.) Alternativ zu `diff` können Sie auch `tkdiff` verwenden, dann erscheint der Vergleich übersichtlicher mit Farbmarkierungen in einem Fenster.

20. Das Ridiculously Simple Filesystem (RSF)

Im Unterordner `mkfs.rfs/` liegt ein kleines Tool, das Sie dort mit `make` übersetzen können. Es erzeugt Dateisystem-Images im RSF-Format (Ridiculously Simple Filesystem). RSF arbeitet mit einer Standardsektorgröße von 512 Byte, und ein Image hat folgenden Aufbau:

- Sektor 0 enthält die FAT (File Allocation Table), die bis zu 32 FAT-Einträge der Größe 16 Byte enthält (16 x 32 = 512).

- Jeder FAT-Eintrag enthält die folgenden Daten:
  - Bytes 0–11: Dateiname. Wenn der Dateiname weniger als 12 Zeichen lang ist, enthalten die restlichen Bytes `\0`. (Achtung: Bei Dateinamen der Länge 12 gibt es hier keine `\0`-Terminierung des Namens.)
  - Bytes 12–13: Dateigröße in Byte, die maximale Dateigröße auf einem RSF-Medium ist also 2^{16} Byte = 64 KByte.
  - Bytes 14–15: Sektornummer des ersten Datenblocks der Datei
- Alle Dateien müssen zusammenhängend im Image gespeichert werden, weil in der FAT nur der Startsektor vermerkt wird. Die erste Datei beginnt in Sektor 1, gleich hinter der FAT. Unbenutzte FAT-Einträge erkennen Sie daran, dass das erste Byte des Eintrags `\0` ist. Hinter einem unbenutzten Eintrag dürfen keine benutzten mehr folgen.

Sie rufen das Tool mit einem Kommando der Form

`./mkfs.rfs hda.img file1 file2 file3` …

auf, um die Dateien `file1`, `file2`, `file3`, … ins Image `hda.img` zu kopieren. Alle Quelldateien müssen im aktuellen Verzeichnis liegen (das Tool kommt nicht mit Pfadangaben zurecht); die Zieldatei (im Beispiel `hda.img`) darf aber auch in einem anderen Ordner liegen.

Sie werden nun Ulix um einen RSF-Treiber erweitern. Entwickeln Sie dazu die folgenden Funktionen, die alle davon ausgehen, dass die zweite virtuelle Platte (`hdb.img`) ein mit `mkfs.rsf` erzeugtes RSF-Image enthält:

- `void rsf_ls()`: Gibt eine Liste aller Dateien im folgenden Format (Größenangabe in Byte) aus

```
filename       size  sectors
test.txt         30  0001-0001
test2.txt     32768  0002-0065
Makefile         12  0066-0066
```

- `int rsf_open(char *filename)`: eine vereinfachte Version des Datei-Öffnens. Es kann immer maximal eine Datei geöffnet sein. Je nach Situation reagiert `rsf_open` wie folgt:
  - Es ist bereits eine Datei geöffnet → Abbruch, Rückgabewert -1.
  - Es ist noch keine Datei geöffnet, aber `filename` existiert nicht auf dem Datenträger → Abbruch, Rückgabewert -1.
  - Es ist noch keine Datei geöffnet, und `filename` existiert → Erfolg. Die Funktion merkt sich in einer globalen Variable den Namen der Datei. Rückgabewert 0.
- `int rsf_readsector(int fd, int secno, char *buf)`: liest einen Sektor aus der geöffneten Datei (falls `fd == 0`). Wenn keine Datei geöffnet ist oder `fd != 0` gilt, Abbruch mit Rückgabewert -1. Ansonsten: Rückgabewert = Anzahl der gelesenen Bytes. Vorsicht: Beim letzten Sektor einer Datei ist das oft ein Wert < 512.
- `void rsf_close(int fd)`: schließt die Datei (falls eine offen war); `fd` ist hier beliebig.

Es gibt keine Funktion zum Schreiben, weil RSF ein Read-only-Dateisystem ist, vergleichbar mit dem ISO-Dateisystem auf CDs und DVDs.

Testen Sie die Funktionen, indem Sie mit `mkfs.rfs` geeignete Images mit Testdateien erstellen und diese dann in Ulix ausgeben lassen.

Wenn Sie noch Zeit haben: Erweitern Sie die Funktionen so, dass Sie mehrere Dateien gleichzeitig öffnen können; der Rückgabewert von `rsf_open()` ist dann ein echter File-Descriptor `fd`.