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.

8. Paging / Literate Programming

Im Ordner `tutorial03/` in Ihrem Home-Verzeichnis finden Sie eine Version des Ulix-Kernels, die (wie in der Vorlesung gezeigt) Paging beherrscht. Sie liegt als Literate Program (`ulix.nw`) vor.

a) Wechseln Sie in den Ordner und extrahieren Sie aus dem Literate Program `ulix.nw` mit `make` die Sourcecode-Dateien `ulix.c` und `start.asm`; diese werden dann auch automatisch übersetzt. Starten Sie dann den Kernel mit `make run`. Das System führt einige Tests der Speicherverwaltung durch und hält dann an.

Werfen Sie auch einen Blick in die PDF-Datei `ulix.pdf` (die Sie nach Änderungen an `ulix.nw` mit `make pdf` neu erzeugen können). Sie finden hier u. a. die Beschreibung des bisherigen Codes für die Rahmen- und Seitenverwaltung und damit auch die ausführliche Variante einer Musterlösung zu Aufgabe 7 vom letzten Übungsblatt.

Es ist nicht nötig, das recht lange Dokument Zeile für Zeile zu lesen, aber Sie sollten es überfliegen und die Art und Weise, wie der Code in verschiedene Code Chunks unterteilt ist, mit Ihrer eigenen Lösung vergleichen.

b) Wir haben in der Vorlesung die Page Table Descriptors und die Page Descriptors als Strukturen (`struct`) definiert und bereits besprochen, dass alternativ eine Interpretation als `unsigned int` denkbar ist. In dieser Aufgabe stellen Sie das Literate Program um, so dass es mit diesen einfachen Integer-Typen arbeitet. Arbeiten Sie in einer Kopie (damit die Originaldateien erhalten bleiben); dazu erzeugen Sie im Home-Verzeichnis mit

```
cp -r tutorial03 tutorial03-copy
```

eine Kopie des ganzen Verzeichnisses und nehmen die Änderungen in `tutorial03-copy/` vor.

Ein paar Tipps zur Vorgehensweise:

(i) Ändern Sie zunächst die Typdeklarationen für `page_table_desc` und `page_desc` in

```
typedef unsigned int page_table_desc;
typedef unsigned int page_desc;
```

(ii) Auf die Typen `page_directory` und `page_table` können Sie ganz verzichten und stattdessen einzelne Verzeichnisse bzw. Tabellen als

```
page_table_desc pd[1024] __attribute__ ((aligned (4096)));
page_desc       pt[1024] __attribute__ ((aligned (4096)));
```

deklarieren. Damit greifen Sie künftig etwa auf den Eintrag `n` von `pt` über `pt[n]` statt `pt.pds[n]` zu. Mit `pt` (ohne Index) steht Ihnen ein Zeiger (vom Typ `unsigned int*`) auf den Anfang der Tabelle zur Verfügung. Um einen einzelnen Deskriptor mit Funktionen wie `fill_page_desc()` oder `fill_page_table_desc()` aufrzurufen, übergeben Sie einen Zeiger auf `pt[n]`, also `&pt[n]`.

(iii) Nach den Änderungen funktionieren alle Funktionen nicht mehr, die mit diesen Typen arbeiten; Sie müssen also jeweils Anpassungen vornehmen. Um z. B. in `fill_page_desc()` einen Page Descriptor mit Inhalt zu füllen, können Sie zunächst den Adresswert `frame_addr` nehmen und dessen niedrigste zwölf Bits auf `0` setzen:

```
tmpvalue = frame_addr & 0b11111111111111111111000000000000;
```

bzw.

```
tmpvalue = frame_addr & 0xFFFFF000;
```

(In der Hexadezimaldarstellung fassen Sie jeweils vier Bits in einer Hex-Ziffer zusammen.) Danach müssen Sie die zu setzenden Bits ergänzen. Sie könnten dazu, auf Basis der Bitpositionen, Flag-Konstanten definieren, etwa wie folgt:

```
#define FLAG_PRESENT  1<<0   //  Bit 0: present
#define FLAG_ACCESSED 1<<5   //  Bit 5: accessed
```

usw. Dann können Sie bei Bedarf über eine „Oder"-Operation (`|`) mit z. B.

```
tmpvalue = tmpvalue | FLAG_ACCESSED;
```

das jeweilige Bit setzen. Wenn alles fertig ist, tragen Sie den Wert mit `*desc = tmpvalue;` ein. (Sie müssen den Funktionsprototyp anpassen und später einen Pointer to `unsigned int` übergeben.)

(iv) Um später aus einem Deskriptor die Adresse zu extrahieren, setzen Sie wieder (wie oben) die untersten zwölf Bits auf `0`. Um hingegen ein einzelnes Flag zu extrahieren, können Sie den Deskriptor mit `FLAG_*` ver-„unden" (`&`) und das Ergebnis auf `==0` bzw. `!=0` testen:

```
if ((descriptor & FLAG_ACCESSED) == 0) { /* flag is not set */ }
```

c) Testen Sie, dass das alte und das neue Programm identisch arbeiten. Dazu können Sie in beiden Programmen die Funktion `hexdump()` verwenden, die einen Adressbereich als Hex-Dump ausgibt. Als Argumente verwenden Sie die Anfangs- und Endadressen der Seitentabelle bzw. des Seitenverzeichnisses, z. B. in der Form

```
hexdump ( (unsigned int)current_pd, (unsigned int)current_pd + 4096 );
```

Bei den Programmaufrufen landen die Ausgaben von `hexdump()` in Dateien `output.txt` (im jeweiligen Ordner), Sie können diese dann später vergleichen:

```
cd ~/tutorial03/;       make;  make run > output.txt
cd ~/tutorial03-copy/;  make;  make run > output.txt
cd ~/;  diff ~/tutorial03*/output.txt
```

Sie sollten beim `diff`-Aufruf keine Ausgabe erhalten: Dann sind die beiden Dateien identisch.

(Die `hexdump`-Aufrufe müssen *nach* den Änderungen an den Tabellen stattfinden.)

d) Prüfen Sie mit `make pdf`, dass Sie aus dem Literate Program auch nach Ihren Änderungen noch eine PDF-Datei erstellen können – falls das nicht klappt, gehen Sie auf Fehlersuche.