1 Übersicht

In Aufgabe 4 soll die bislang entwickelte virtuelle Maschine um die Möglichkeit des Paging erweitert werden.

2 Organisatorisches

Diese Aufgabe ist optional. Wenn Sie wollen, können Sie Ihre Lösung per Mail senden an i4vm@cs.fau.de. Die Projektdateien mit Test-Programmen finden Sie unter https://www4.cs.fau.de/Lehre/WS14/V_VM/Uebungen/aufgabe4.tar.gz.

3 CPU-Simulator

3.1 Erweiterung um Paging

Um Paging nutzen zu können, muss der CPU-Simulator um die Steuerregister `cr0`, `cr2` und `cr3` erweitert werden. Von `cr0` braucht hierbei lediglich Bit 31 berücksichtigt werden, welches Paging aktiviert beziehungsweise deaktiviert. Tritt ein Seitenfehler auf, so soll die linear Adresse, welche den Fehler hervorgerufen hat, in dem Register `cr2` hinterlegt werden. Schließlich ist die Basisadresse des Seitentabellenverzeichnisses in dem Register `cr3` festgelegt.

Die `x86`-Architekur erlaubt eine Vielzahl an Möglichkeiten, um Paging nutzen zu können. Die Test-Programme dieser Aufgabe beschränken sich jedoch darauf, 4 KiB Seiten mit einer zweistufigen Seitentabellenhierarchie zu verwenden. Außerdem braucht der Einfachheit halber lediglich der Zugriff auf eine Seite, welche als nicht present markiert ist, zu einem Seitenfehler führen.

3.2 Benötigte Opcodes

In den Testprogrammen dieser Aufgabe befinden sich folgende neue Opcodes:

- `invlpg`
- `mov` (von/nach Control Register)
- `or`
- `shr`

3.3 Translation Lookaside Buffer

Bei Verwendung von *Paging* muss bei jedem Speicherzugriff die physikalische Adresse anhand der Seitentabellen berechnet werden. Dies bedeutet, dass zusätzliche Speicherzugriffe getätigt werden müssen. Aus Performanzgründen besitzt eine CPU daher einen *Translation Lookaside Buffer*, welcher die Zuordnungen von linearer zu physikalischer Adresse zwischenspeichert. Können Sie Ihre virtuelle Maschine durch Implementierung eines *TLB* beschleunigen?

4 Testprogramm

Das Testprogramm finden Sie im Verzeichnis `test-paging`. Benötigte Bibliotheken zur Ein-/Ausgabe liegen in den Verzeichnissen `lib_real` bzw. `lib_vm`. `lib_real` enthält

hierbei Varianten, die mit real existierender Hardware bzw. mit dem Linux-Kern kommunizieren können, `lib_vm` die entsprechenden Implementierungen für die vereinfachte VM.

Verzeichnis `test-paging`:

- `bios_boot.S`: Implementierung eines erweiterten Boot-Loaders, welcher die Segmentregister initialisiert und die ersten 24 Sektoren der Festplatte („das Betriebssystem“) in den Hauptspeicher ab Adresse `0x0000` kopiert. Nach dem Kopieren wird an Adresse `0x0000` gesprungen.

- `main.S`: „Betriebssystem“, welches die Hardwarekomponenten initialisiert und ein Menü anzeigt, mit dem jeweils ein Prozess gestartet werden kann. Jeder auswählbare Prozess wird hierbei zunächst von Festplatte an die physikalische Adresse `0x5000` (Code) und `0x6000` (Daten) kopiert. Danach wird ab Adresse `0x3000` das Seitentabellenverzeichnis und ab Adresse `0x4000` das Seitenverzeichnis für den Prozess generiert. Schließlich wird an die lineare Adresse `0x01000000` gesprungen, welche auf die physikalische Adresse `0x5000` abgebildet wird. Tritt ein Seitenfehler auf, so wird der Prozess mit Fehlermeldung beendet.

  Der Prozess 2 besteht aus zwei Seiten an Prozessdaten. Tritt an einer Adresse dieser Datenseiten ein Seitenfehler auf, so wird die aktuell an Adresse `0x6000` geladene Seite auf Festplatte geschrieben und die entsprechend fehlende Seite eingelagert, und anschließend die Seiteneinträge aktualisiert. Achtung: Die ausgelagerte Datenseite wird an dieselbe Position geschrieben, an der die ursprünglichen Daten des Programms lagen. Gegebenenfalls müssen Sie das Festplattenimage löschen und neu erzeugen, um die ursprünglichen Daten zurückzuerhalten.

  Der Menüpunkt 3 versucht von Adresse `0x12345678` zu lesen, und sollte somit sofort einen Seitenfehler erzeugen.

- `hallo.S`: „Hallo Welt“ als Prozess.

- `arith.S`: Berechnen der Fakultät als Prozess.

- `Makefile`: Baut sowohl ein BIOS Rom mit dem Bootloader also auch ein Festplatten-Image.

- `page_maps.txt`: Übersicht über den physikalischen Adressraum und den virtuellen Adressraum eines Prozesses.

- Unterverzeichnis `x86_cdrom`: Das dortige `Makefile` erzeugt ein CD-ISO-Image, welches die Testprogramme mit den Bibliotheken für echte Hardware als Bootloader enthält. Somit sollte es auf einem realen System funktionieren.