Heinrich Heine
Universität Düsseldorf

UNIVERSITÄT DÜSSELDORF
Mathematisch-Naturwissenschaftliche Fakultät
Institut für Informatik

Prof. Dr. Michael Schöttner
Dr. Fabian Ruhland

3. Aufgabenblatt zum Modul „Isolation und Schutz in Betriebssystemen“

Alle Materialien finden Sie in ILIAS

Lernziel

Das Ziel dieser Aufgabe ist es, eine physikalische Speicherverwaltung für Kacheln (engl. page frames) zu implementieren und den Speicher für den Kernel-Heap darüber zu allozieren. Die physikalische Speicherverwaltung ist die Grundlage für das Paging, welches in Aufgabenblatt 4 folgt.

Aufgabe 1: Verfügbaren physikalischen Speicher ermitteln

Bevor das Paging implementiert wird, muss zunächst ermittelt werden wie viel und wo nutzbarer physikalischer Speicher vorhanden ist. Die Lösung ist in `multiboot.rs` in den Funktionen `get_free_memory` und `check_for_overlapping` bereits vorhanden, soll aber gelesen und verstanden werden.

Die notwendigen Informationen bekommen wir von Multiboot, in Form von `mmap`-Einträgen. Die `mmap`-Regionen können sich überlappen oder sich sogar widersprechen. In solchen Fällen gehen wir defensiv vor, behandeln solche Speicherbereiche als reserviert. Außerdem sollte der Speicher bis zum ersten Megabyte ignoriert werden sowie der Bereich von 15 MiB bis 16 MiB („ISA memory hole“).

Unser Kernel-Image wird an die Adresse 1 MiB geladen, jedoch wird der durch das Kernel-Image belegte Speicherbereich nicht in den `mmap`-Einträgen von Multiboot berücksichtigt. Deswegen wird dieser Speicherbereich in `kmain` anhand `___KERNEL_START___` und `___KERNEL_END___` berücksichtigt. Diese beiden „Labels“ sind in der Linker-Datei definiert, im Ordner `boot`.

Folgende Dateien sind für diese Aufgabe relevant: `multiboot.rs, const.rs, startup.rs`

Information zu Multiboot, insbesondere den mmap-Einträgen finden Sie hier:
https://www.gnu.org/software/grub/manual/multiboot/multiboot.html

Aufgabe 2: Ein Allokator für Page-Frames

Der verfügbare physikalische Speicher muss in 4 KiB Page_Frames verwaltet werden. Hierfür wird nun ein Page-Frame-Allokator benötigt, als Basis empfiehlt sich der vorhandene Heap-Allokator (in `list.rs`), der Code muss natürlich angepasst werden. Wir verketten also die freien Page-Frames, wobei ein Block aus mehreren aufeinanderfolgenden Page-Frames bestehen kann. Die Metadaten für frei Blöcke schreiben wir direkt in die freien Page-Frames. Für die belegten Page-Frames benötigen wir keine Metadaten, da es nur 4 KB Page-Frames gibt und jede Page-Frame von einem Seitentabellen-Eintrag referenziert wird.

Bei der Initialisierung müssen wir die Informationen bzgl. des freien Speichers von der letzten Aufgabe verwenden. Hier muss unbedingt auf die 4 KB Alignierung geachtet werden und notfalls konservativ zugeschnitten werden. Im einfachsten Fall hat der Allokator nach der Initialisierung nur einen Eintrag in der Liste, welcher alle Page-Frames umfasst. Es können aber mehrere sein, je nachdem welche Informationen wir von Mutiboot erhalten haben. Die Page-Frame-Blöcke sollten in der Liste sortiert sein, nach aufsteigenden Startadressen.

Es soll möglich sein eine oder mehrere aufeinanderfolgende Page-Frames zu allozieren bzw. freizugeben. Beim Freigeben müssen die Blöcke verschmolzen werden!

Wenn der Allokator erfolgreich getestet wurde, sollen zwei globale Allokatoren realisiert werden, einen für Kernel-Page-Frames: 0 – 64 MiB - 1 und einen für User-Page-Frames 64 MiB – Ende.

In folgenden Dateien muss gearbeitet werden: `frames.rs`

Wichtige Hinweise:

- *Achtung: Der Allocator darf keinesfalls einen reservierten Page-Frame erneut vergeben! Bevor Sie fortfahren, sollten Sie die Korrektheit Ihres Page-Frame-Allokators testen!* `assert` *ist hier sehr nützlich.*
- *Ferner empfiehlt es sich Page-Frames beim Allozieren mit 0 zu initialisieren. So können später Pointer-Fehler einfacher erkannt werden, da dann ein Null-Pointer-Zugriff mithilfe des Pagings erkannt werden kann.*
- *Schließlich sollte in Qemu die DRAM-Speichergröße auf mindestens 256 MB eingestellt werden, Default sind 128 MB*

Aufgabe 3: Kernel-Heap

Nun soll noch die Initialisierung des bestehenden Allokator in `startup.rs` angepasst werden. Statt an einer festen Adresse einfach freien Speicher zu vermuten, soll nun freier Speicher vom Page-Frame-Allokator (siehe Aufgabe 2) für den Heap angefordert werden, beispielsweise 1 MB, siehe auch Konstanten in `consts.rs`. Nun sollten alles weiterhin funktionieren.

Die Stacks werden vorerst noch im Kernel-Heap alloziert. Dies ändert sich mit dem nächsten Aufgabenblatt, wenn wir Paging einführen.

Hinweis: Adressraumaufteilung

Der physische Adressraum wird in unserem Betriebssystem fest aufgeteilt, 0 bis 64 MiB – 1 für den Kernel und der Rest für den User-Mode. Auch dies dient dazu die Entwicklung und das Debugging zu vereinfachen. Diese Aufteilungen ist durch Konstanten in `consts.rs` definiert.

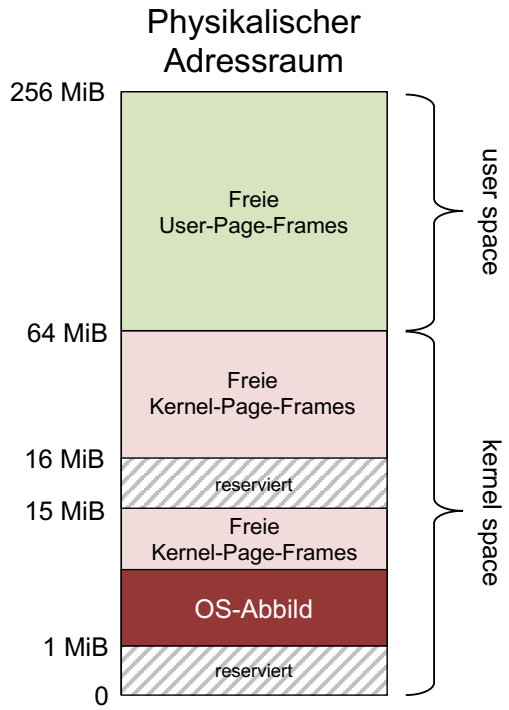

Abbildung 1, Aufteilung des physikalischen Adressraumes