Adressräume & Windows Memory Management

Prof. Dr. Sharam Gharaei

Version 1.2.0,
07.04.2017

Inhaltsverzeichnis

1 Einleitung

Ein Adressraum ist ein adressierbarer Bereich des Speichers. Zum Beispiel verfügt ein Prozessor mit einem 32 Bit Adressbus über einen Adressraum von 4 GBytes (2^{32} Bytes) in seinem physischen Speicher. Dieser Adressraum wird als **flat** bzw. nicht differenziert bezeichnet. Dabei sind die Adressen der Speichereinheiten (Bytes) fortlaufend.

In den meisten Betriebssystemen wird eine strikte Trennung zwischen der Behandlung der User-Prozesse und Kernel-Prozesse vorgenommen.

Windows benutzt eine elementare Methode, um diese zu realisieren. Dafür wird der physische Speicher in zwei Adressräume aufgeteilt:

- User-Adressraum
- Kernel-Adressraum

Die Segregation wandelt den flachen Adressraum in differenzierte Adressräume um und hält somit User-Prozesse von Kernel-Prozessen separat. Die Differenzierung bezieht sich auf drei Aspekte:

1. Gespeicherte Daten in einem Adressraum
2. Identifikation der Speichereinheiten (idR Bytes) in einem Adressraum durch Adressierung
3. Zugriff auf Speichereinheiten in einem Adressraum

Während die Relevanz der beiden ersten Themen aus dieser Liste, d.h. Inhalt und die Adresse einer Speichereinheit, intuitiv erkennbar sind, bedarf der dritte Punkt einer Erläuterung. Wenn ein User-Prozess bestimmte Funktionen vom Betriebssystem benötigt, z. B. Zugriff auf eine Datei, ruft er eine entsprechende Kernel-Funktion auf. Diese Operation wird als **System Call** bezeichnet.

Im User-Adressraum sind für die User-Applications die OS-Funktionen lediglich über **System Call Interface** (Subsystem-DLLs) erreichbar. Dafür benutzt eine User-App das Windows-API. Das System Call Interface umfasst Code-Bibliotheken (DLLs). Seine Rolle besteht darin, eine dokumentierte Funktion aus Windows-API in die entsprechende Funktion für den Kernel-Adressraum zu übersetzen[1]. Durch diese zweischichtigen API-Aufrufe wird ein direkter Zugriff auf Kernel noch einmal verhindert.

Eine Trennung der Adressräume bezieht sich damit nicht nur auf die Speicherarchitektur, sondern auch auf die Ausführung des Codes. D.h. die Daten einer User-Application gehören zum User-Adressraum, während die Daten der Prozesse vom OS-Kernel sich im Kernel-Adressraum befinden. Auf der anderen Seite gehören die Instruktionen (Anweisungen) einer User-App zum User-Adressraum, während die Kernel-Funktionen im Kernel-Adressraum ausgeführt werden. Diese einfache Differenzierung wird im Folgenden noch präziser dargelegt.

[1] Die Funktionen im Kernel-Adressraum sind häufig nicht dokumentiert

1.1 Code-bezogene Aspekte

Die Code-bezogenen Aspekte eines Adressraums sind:

- Im Kontext der Ausführung einer Instruktion kann ein Prozess auf Speicher zugreifen. Daher soll generell die Adressraum-Zugehörigkeit pro Instruktion festgelegt werden. Dies wird bestimmen, welche Instruktionen durch User- und welche ausschließlich durch Kernel-Prozesse ausgeführt werden dürfen.
- Ebenso soll unabhängig von einer Ausführung der Zugriff auf Speicher kontrolliert werden, d.h. welche Prozesse auf welche Speicheradressen zugreifen dürfen.

1.2 Speicherungsbezogene Aspekte

Die Speicherungsbezogenen Aspekte umfassen folgende Fragen:

- Wo im physischen Speicher dürfen die Daten der User-Apps gespeichert werden?
- Wo im physischen Speicher dürfen die Daten des OS-Kernels gespeichert werden?

2 Grundlage der Realisierung

Nicht die Segregation der Ebenen, sondern die Detail der Realisierung sind für die Anfälligkeit bzw. die Robustheit der Architektur eines OS verantwortlich. Vor allem ist dafür eine direkte Unterstützung durch den Prozessor (die CPU) erforderlich. Das Grundkonzept hierfür ist relativ einfach zu erfassen: Jede Instruktion (vgl. 1.1) und jede Adresse (vgl. 1.2) wird mit Zusatzinformationen versehen, damit diese eindeutig zu einem Adressraum zugeordnet werden. Dies kann in einfachster Form mit Hilfe von einigen Extra-Bits realisiert werden.

Ein entscheidendes Merkmal der Windows-Architektur ist, dass sie sich stark auf die Gegebenheiten und Restriktionen der Intel-Hardware bezieht.[2] Die IA-32 Architektur[3] bietet einige Techniken an, die eine Segregation der Adressräume auf der OS-Ebene, als Bestandteile der Speicherverwaltung, direkt unterstützen. Die wichtigsten davon sind:

- Ein durchgehend forciertes Berechtigungsschema durch Privileg-Ringe
- Segmentierung des physischen Speichers

Eine wichtige Begleiterscheinung dabei ist hier anzumerken. Die Erfahrungswerte deuten darauf hin, dass diese Mechanismen einen hohen Grad an Widerstand gegen gezielte Manipulationen der Kernelprozesse aufweisen.

Es wird darauf hingewiesen, dass viele Erkenntnisse, die aus folgenden Unterthemen gewonnen werden, gänzlich bzw. teilweise auf andere Architekturen, z.B. AMD bzw. auf andere Betriebssysteme, z.B. Linux, übertragen werden können.

[2] Diese Tatsache wird zwar von Microsoft dementiert, trotzdem gibt es viele Anhaltspunkte, die sie bestätigen.
[3] IA-32 steht für die "**32**-bit **I**ntel **A**rchitecture" und bezieht sich auf Systeme auf Basis von 32-Bit-Prozessoren. Die IA-32 Intel-Prozessoren sind mit Intel Pentium II kompatibel. Die CPUs anderer Hersteller werden ebenso als IA-32 bezeichnet, falls sie die den gleichen Befehlssatz sowie ein 32-Bit-Betriebssystem unterstützen.

3 Die Realisierung

3.1 Die Privileg-Ringe

Laut Intel (1) benutzen sowohl die IA-32-Architektur, inklusive Intel 64[4], als auch IA-64-Prozessoren (Itanium) ein vier schichtiges Berechtigungsschema (*privelege layers*), das aus vier umschließenden Ringen besteht. Der erste davon wird als Ring 0 bezeichnet. Die anderen werden abwärts als Ring 1, 2 und drei bezeichnet. Diese vier Stufen werden dem Speicher zugeordnet. Damit bestimmen sie, welche Prozesse auf den Speicher zugreifen dürfen.

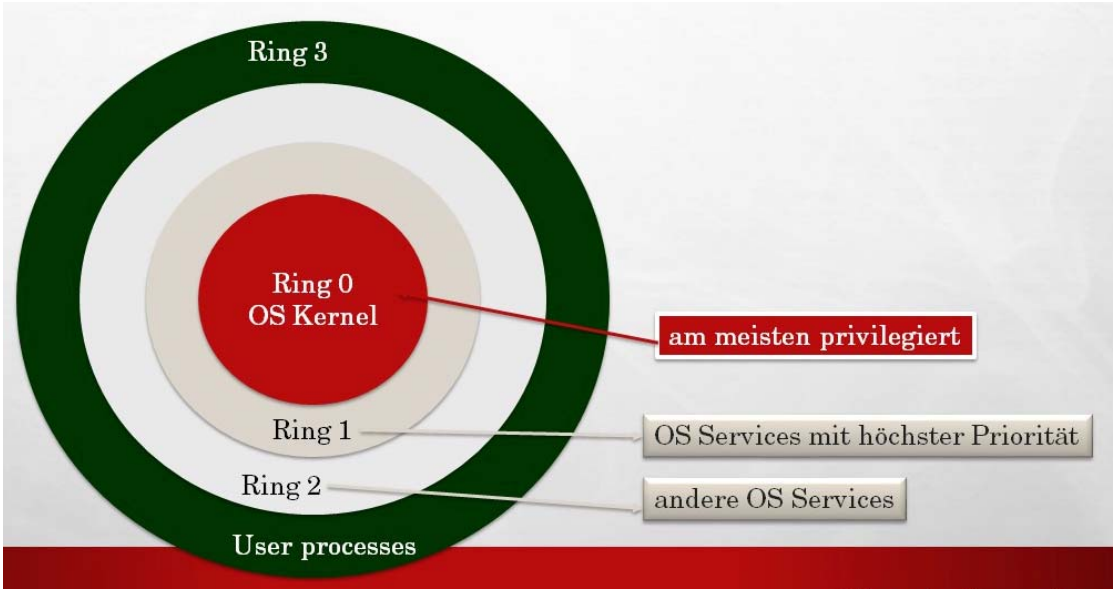

Abbildung 1 Die Privileg Ringe

Wie in **Error! Reference source not found.** dargestellt, sieht das Schema folgende Berechtigungen vor:

- Ring 0: OS Kernel
- Ring 1: OS Services mit höchstem Privileg
- Ring 2: andere OS Services
- Ring 3: User Prozesse

Damit können zwei Bits die Zugehörigkeit einer Instruktion bzw. einer Speicheradresse zu einem Ring darstellen. Diese bezeichnen wir im Folgenden als Ring-Bits. Konkret können Sie für folgende Zwecke eingesetzt werden:

- Die Ring-Bits in Anweisungen können benutzt werden, um die CPU-Funktionalität abhängig von dem tatsächlichen Privileg zu steuern.
- Die Bits steuern den Zugriff auf Prozess-Adressräume im Speicher.

[4] CPUs, die auf IA-32 basieren und eine Erweiterung für Unterstützung der 64-Bit-Architektur aufweisen.

3.2 Segmentierung

Unter der Wintel Architektur existieren drei Formen von Speicheradressen (s.Abbildung 2)

- Die **Physische Adresse** entspricht dem tatsächlichen Adressierungsschema der Hardware.
- Die **Lineare Adresse** ähnelt stark dem physischen Adressraum. Aus prozessorspezifischen Gründen kann jedoch die Anfangsadresse des linearen Adressraumes von der ersten Adresse des physischen Adressraums abweichen.
- Die **Segmentbasierte Adresse** ist eine abstrakte Adresse, die aus zwei Bestandteilen (a) Segmentadresse und (b) Offset-Adresse innerhalb des Segments identifiziert wird. Dabei wird der verfügbare Speicher in Abschnitte aufgeteilt, die Segmente genannt werden. Die Segmente dürfen unterschiedliche Größen haben. In diesem Adressierungsschema wird eine Adresse generell als ***logische Adresse*** bezeichnet.

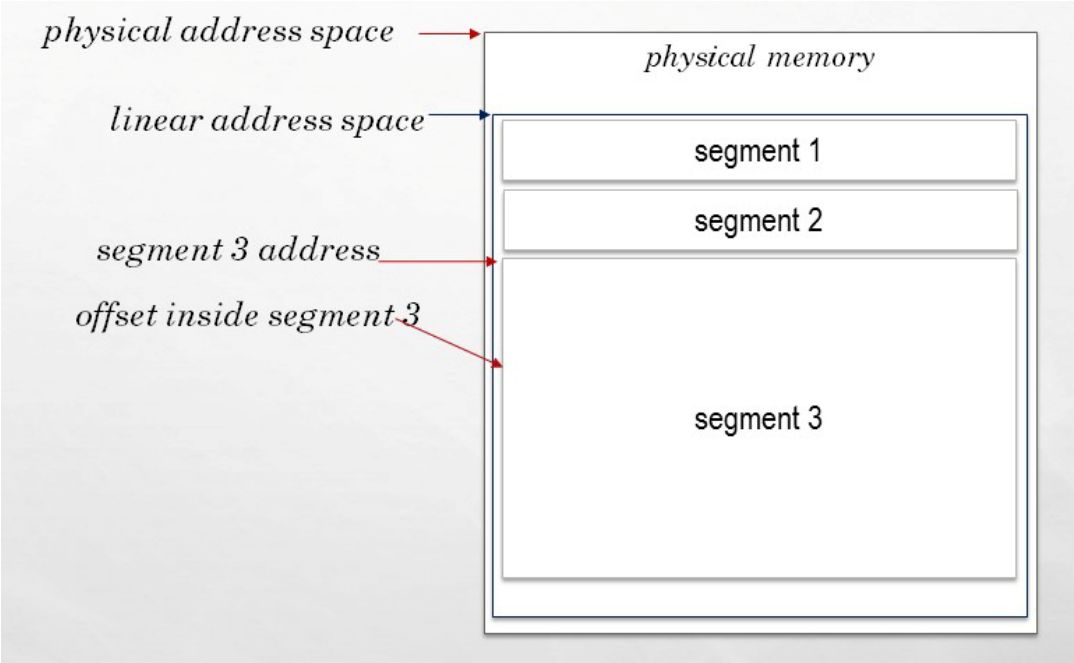

Abbildung 2 Drei Formen von Speicheradressen

Segmentierung ermöglicht eine *Isolierung* der individuellen Codes und Daten einzelner Prozesse. Somit können mehrere Prozesse auf dem gleichen Prozessor laufen, ohne sich gegenseitig zu stören. Für diesen Zweck weist das OS jedem Prozess während seiner Ausführung dedizierte Segmente zu. Dafür wird ein Mechanismus verwendet, der von mehreren Prozessorregistern unterstützt wird.

Abbildung 3 präsentiert einen Überblick über die Wintel-Segmentierung. Hier wird die logische Adresse als Kombination von einem 16-Bit **Segment-Selektor** und einer 32-Bit

Offset-Adresse dargestellt. Anhand von diesen Bestandteilen wird eine konkrete Adresse aus dem linearen Adressraum berechnet.

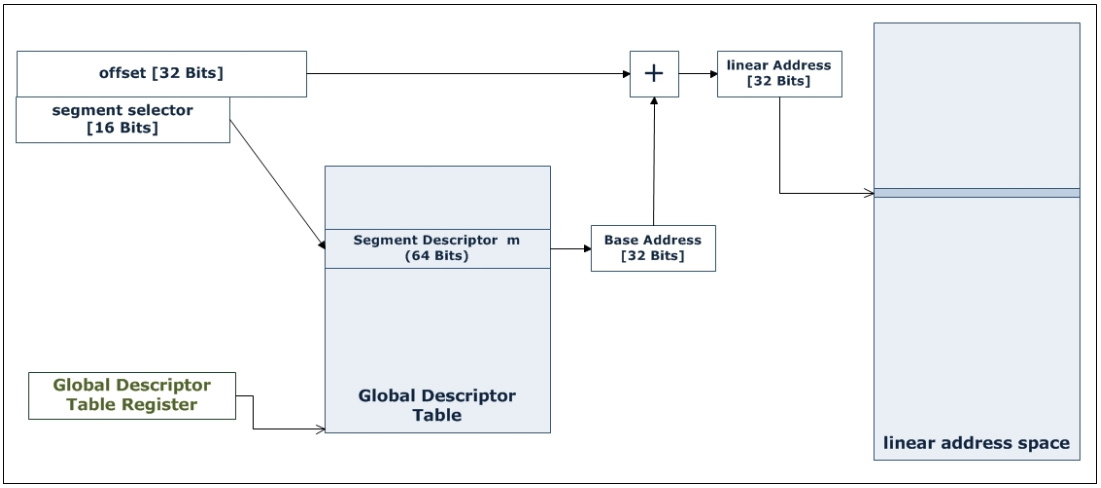

Abbildung 3 Die Segmentierung in der Wintel Architektur

Der Segmentselektor ist selbst ein Zeiger auf eine 64-Bit Struktur innerhalb von einer Tabelle, die als **Global Descriptor Table (GDT)** bezeichnet wird. Diese Tabelle gehört der CPU und enthält eine Liste aller Segmente, die systemweit verfügbar sind. Jeder Eintrag in dieser Liste ist ein **Segment Descriptor** für ein Segment im *linearen* Adressraum. Der Descriptor besteht aus einer Struktur, die u.a. folgende Daten trägt:

- Die **Segment Basis-Adresse** repräsentiert die Adresse vom ersten Byte des Segments im linearen Adressraum.
- Das **Segment-Limit** legt die Größe des Segments fest.
- Der **Segmenttyp**, der entweder als (a) Data Segment, (b) Code Segment oder (c) System-Segment spezifiziert werden kann.
- Die **Berechtigungsschemata** für Segmenttypen werden pro Segmenttyp festgelegt. Beispiele für Berechtigungen sind Lese-Schreibe-Zugriff (Data), Ausführung (Code) und Trap-Handling (System).
- Das **Segment-Privileg** wird als Ring-Bits festgelegt.

Die Segment Descriptors können durch Compiler, Linker, Loader oder Kernel-Executives erstellt werden. User-Applications bekommen dagegen keine direkte Möglichkeit dazu. Wenn ein User-Prozess eine API-Funktion aufruft, die in Ring 1 implementiert wurde, dann muss ein System-Segment mit Ring-Bits-Wert 1 bzw. niedriger im RAM geladen werden. Der User-Prozess kann nicht direkt auf dieses System-Segment zugreifen, während der Kernel freien Zugang dazu hat. Trotzdem hat der User-Prozess damit eine Zugriffsmöglichkeit auf Kernelfunktionen.

3.3 Virtual Memory

Die Intel-Architektur erlaubt die Realisierung von **Virtual Memory** anhand von optionalem *Paging* in Kombination mit Segmentierung (2). Für diesen Zweck wird der Prozessor einen Paging-Mechanismus einsetzen, der den 32-Bit linearen Adressraum in gleich große Blöcke aufteilt, die Pages genannt werden. Diese linearen Adressen werden dann auf den physischen Adressraum abgebildet.

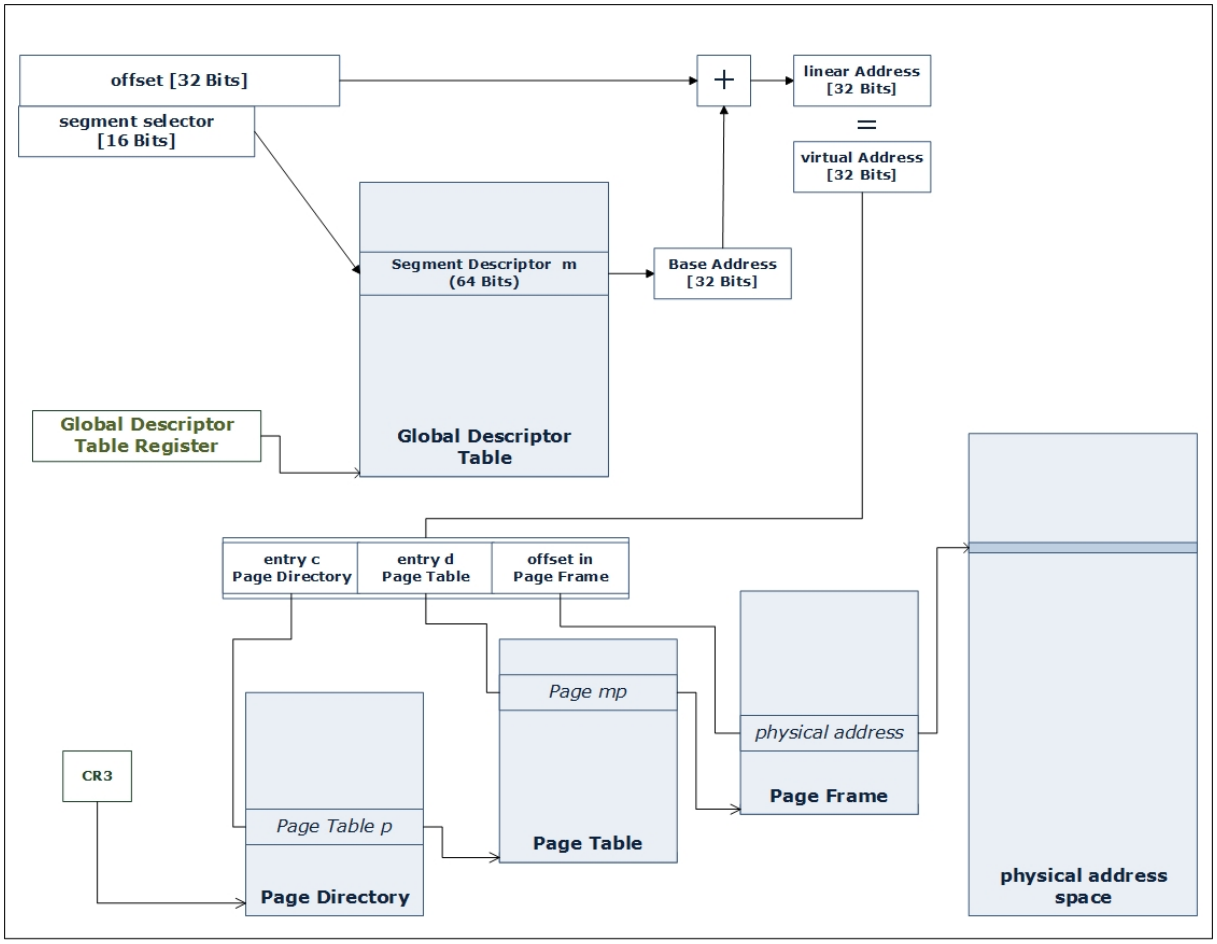

Abbildung 4 Segmentierung mit Paging in Intel A32-Architektur

Eine Page-Adresse ist eine dreiteilige Struktur (siehe Abbildung 4). Sie umfasst u.a. die physische Adresse einer Page, die selbst in einer **Page Table** zu finden ist. Die physische Adresse von der relevanten Page Table ist als Eintrag dem **Page Directory** zu entnehmen. Technisch gesehen sind alle diese drei Komponenten Zeigertabellen und werden daher auch als **Prozeßsegment-Zeigertabellen** bezeichnet. Funktional gesehen können sie wie folgt beschrieben werden:

- *Page Directory* (bits 22-31) bewahrt ein Verzeichnis von allen Gruppen von Pages auf, die als einzelne Page Tables eingetragen werden.
- *Page Table* (bits 12-21) enthält die Page Offsets für eine Gruppe von logisch zusammenhängenden Pages.
- *Page Offset* (bits 0-11) ist ein Zeiger auf die physische Adresse einer Page im

Speicher. Die Größe einer Page ist vordefiniert und ist unveränderbar.

Die physische Adresse von Page Directory wird selbst in einem Register (CR3) aufbewahrt.

Wie wird der Zugriff auf den physischen Adressraum in diesem Schema reguliert? Das Berechtigungsschema, das für Segmentierung ohne Paging gilt, wird bei der Abbildung von der linearen Adresse auf die physische Adresse weitergegeben, indem in den beiden Feldern Page Directory und Page Table ähnliche Bits einen Schutz des Speichers erzwingen.

Literaturverzeichnis

1. **Intel.** http://www.intel.com/technology/itj/2006/v10i3/1-hardware/3-software.htm. [Online] Intel Corp. [Zitat vom: 20. 11 2011.]

2. **Blunden, Reverand Bill.** *The Rootkit Arsenal, Escape and Evasion in the Dark Corners of the System.* s.l. : Wordware Pub Co, 2009. 978-1598220612.