

Betriebssystemtechnik

Adressräume: Trennung, Zugriff, Schutz

IX. Adressraummodelle: Einadressraumsysteme

SS 2026

Wolfgang Schröder-Preikschat / Volkmar Sieh



Lehrstuhl für Informatik 4
Systemsoftware



Friedrich-Alexander-Universität
Technische Fakultät

Einadressraumsystem

Einleitung

Rückblick

Modell

Adressierung

Schutz

Gedankenspiel

Diskurs

Zusammenfassung

- Einadressraumkonzept
 - Betriebssystem und Anwendungsprogramme teilen sich einen Adressraum
 - der logische Adressraum zeichnet sich nicht mehr als Schutzdomäne aus
 - Adressen werden eine bestimmte **Befähigung** (*capability*) zugeschrieben
 - erzeugt, zugeteilt, verwaltet, entzogen, zerstört durch das Betriebssystem
 - Prozesse greifen damit auf *sämtliche* Objekte des Rechensystems zu

- Einadressraumkonzept
- Adressen haben eine eindeutige Interpretation
- Prozesse können alle Daten im System benennen, haben jedoch für gewöhnlich nicht das Recht, auf alle diese Daten zuzugreifen

Eigenschaften von Mehradressraumsystemen, die (total/partiell) private Adressräume implementieren:

- **partielle Virtualisierung** [4, 17] der Prozessadressräume
 - Vergrößerung der verfügbaren Menge von Adressräumen
 - Einrichtung harter Speicherschutzgrenzen
 - Aufräumen, wenn Programme aussteigen, gestaltet sich einfach

Eigenschaften von Mehradressraumsystemen, die (total/partiell) private Adressräume implementieren:

- **partielle Virtualisierung** [4, 17] der Prozessadressräume

- Erschwernis der Kooperation zwischen den Anwendungsprogrammen:
 - Zeiger sind außerhalb der Grenze/Lebenszeit von Prozessen bedeutungslos
 - zeigerbasierte Information mitbenutzen, speichern, übertragen ist schwer
 - der hauptsächliche Kooperationsmechanismus greift auf Kopieren zurück
 - und geht ggf. auch einher mit der Konvertierung in eine kanonische Form

Eigenschaften von Mehradressraumsystemen, die (total/partiell) private Adressräume implementieren:

- **partielle Virtualisierung** [4, 17] der Prozessadressräume

- Erschwernis der Kooperation zwischen den Anwendungsprogrammen:

- Mitbenutzung von Informationen nur auf „Umwegen“ möglich
 - Konsequenz aus der strikten Isolation von Anwendungskomponenten
 - bereits die Gemeinschaftsbibliothek (*shared library*) fällt aus dem Rahmen

Einadressraumsystem

Einleitung

Rückblick

Modell

Adressierung

Schutz

Gedankenspiel

Diskurs

Zusammenfassung

Trennung von Belangen (*separation of concerns* [8, S. 1])

Trennung von Belangen (*separation of concerns* [8, S. 1])

- Adressierung einerseits
 - Adressen sind eindeutig und potentiell für immer gültig
 - darüber hinausgehend sind Adressen kontextunabhängig
 - sie lösen jederzeit zu demselben Datum auf
 - unabhängig davon, welcher Aktivitätsträger sie benutzt/generiert
 - ein Programmfaden kann jedes Datum im System darüber „benennen“

Trennung von Belangen (*separation of concerns* [8, S. 1])

- Adressierung einerseits
 - Adressen sind eindeutig und potentiell für immer gültig

- Schutz andererseits
 - nicht jedes vom Programmfaden adressierte Datum ist von ihm zugreifbar
 - vielmehr definiert die Schutzdomäne eines Fadens seine Zugriffsrechte
 - Begrenzung des Zugriffs/der Zugriffsart auf einen bestimmten Adressbereich
 - permanent oder zeitlich beschränkt
 - Zugriffsrechte ändern sich beim Durchwandern von Schutzdomänen

Trennung von Belangen (*separation of concerns* [8, S. 1])

- Adressierung einerseits
 - Adressen sind eindeutig und potentiell für immer gültig

- Schutz andererseits
 - nicht jedes vom Programmfaden adressierte Datum ist von ihm zugreifbar

↔ seit Multics [6] ist beides (oft) mit dem Begriff „Prozess“ verbunden

Trennung von Belangen (*separation of concerns* [8, S. 1])

- Adressierung einerseits
 - Adressen sind eindeutig und potentiell für immer gültig

- Schutz andererseits
 - nicht jedes vom Programmfaden adressierte Datum ist von ihm zugreifbar

↔ seit Multics [6] ist beides (oft) mit dem Begriff „Prozess“ verbunden —
genau genommen: „Prozessexemplar“

Prozessoren mit breiten Adressen fördern den Einadressraumansatz, da sie den Zwang zur Wiederverwendung von Adressen aufgeben

Prozessoren mit breiten Adressen fördern den Einadressraumansatz, da sie den Zwang zur Wiederverwendung von Adressen aufgeben

- ein solcher Zwang besteht für 32-Bit – und kleineren – Architekturen
- für 64-Bit Architekturen können Adressen auf immer gültig bleiben:

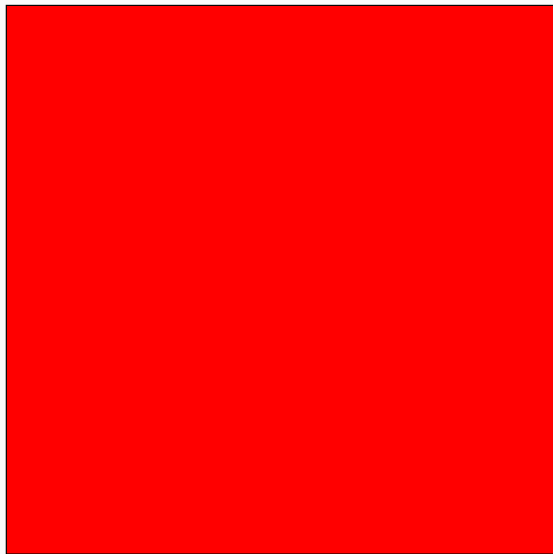
A full 64-bit address space will last for 500 years if allocated at the rate of one gigabyte per second. [3, S. 272]

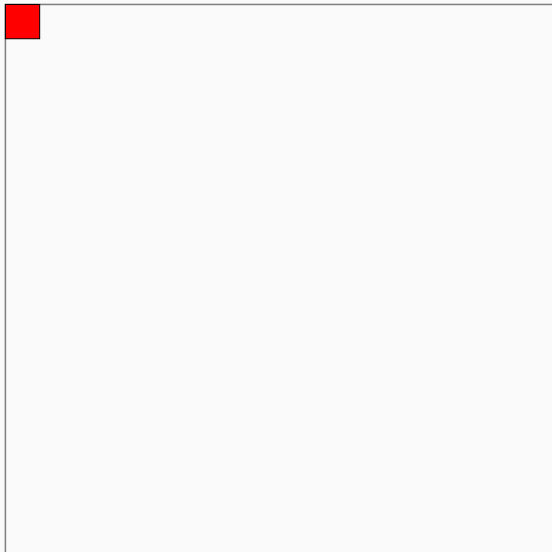
Prozessoren mit breiten Adressen fördern den Einadressraumansatz, da sie den Zwang zur Wiederverwendung von Adressen aufgeben

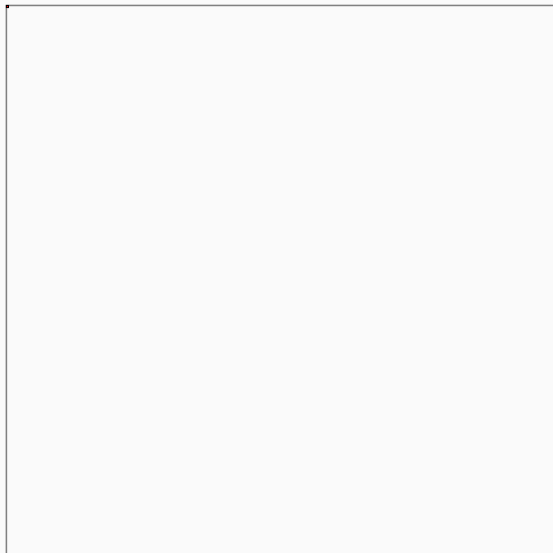
- ausschlaggebend ist der von Programmfäden sichtbare Namensraum
 - gegeben durch die Anzahl der von ihnen technisch aufzählbaren Adressen
 - die tatsächlich in der Hardware implementierte Adressbreite ist belanglos
 - wenn also z.B. nur max. 44-Bit breite reale Adressen möglich sind
 - auch hier ist die Virtualität von 64-Bit Adressen der wesentliche Punkt

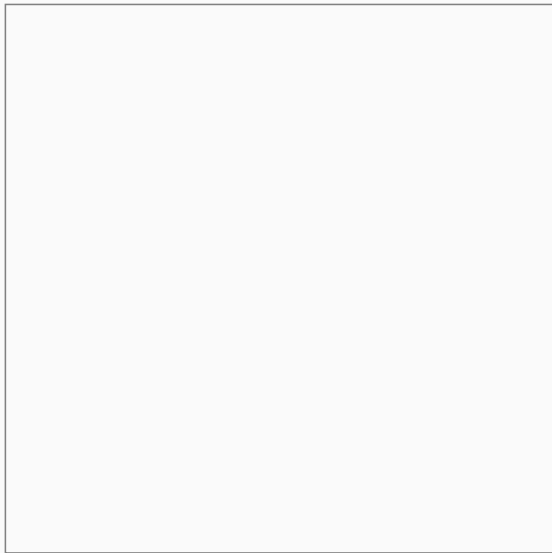
Prozessoren mit breiten Adressen fördern den Einadressraumansatz, da sie den Zwang zur Wiederverwendung von Adressen aufgeben

- ausschlaggebend ist der von Programmfäden sichtbare Namensraum
- der sämtliche im Rechensystem referenzierbare Eintitäten umfasst





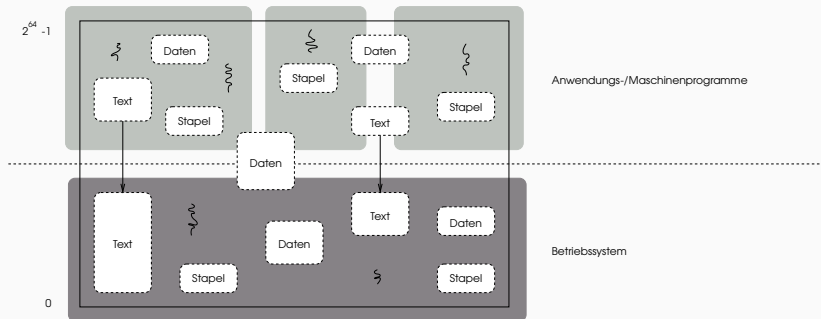




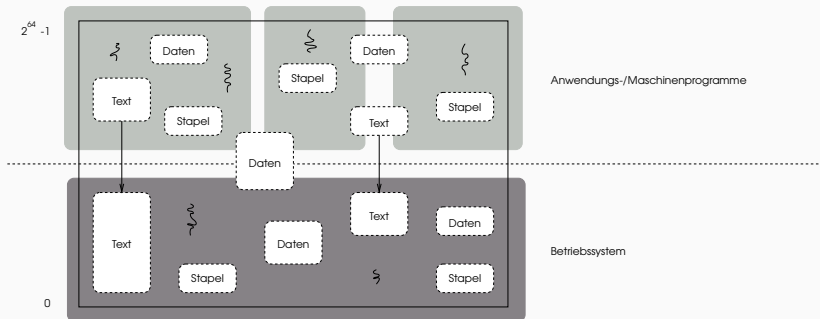
Man könnte den ganzen Hauptspeicher
buchstäblich im 64-Bit-Raum verstecken
und er würde nie gefunden werden.

In einem sehr großen, spärlichen
Adressraum kann die virtuelle Adress-
zuordnung als Befähigung (*capability*)
gelten [24].

„Hyperadressraum“ mit Schutzdomänen



„Hyperadressraum“ mit Schutzdomänen

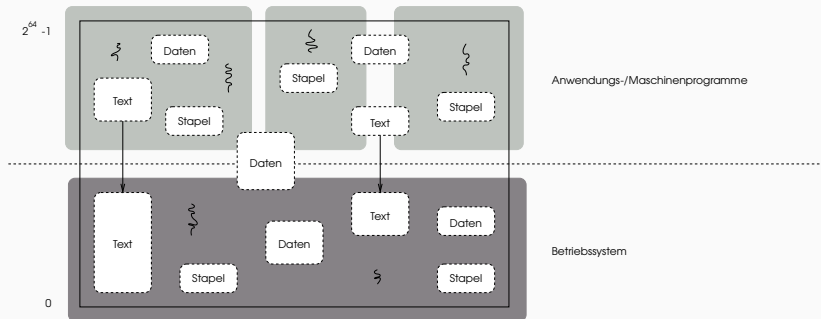


■ horizontaler und vertikaler Informationsaustausch

horizontal
vertikal

- Unterprogrammaufrufe, gemeinsame Adressbereiche
- *ditto*, System-Calls

„Hyperadressraum“ mit Schutzdomänen



- mikrokernbasierte Beispiele von Betriebssystemen der Art:

Opal [3]
Mungi [13]

- Mach, System- und Fernaufrufe [16] über Portale
- L4, Interaktion nur über gemeinsame Adressbereiche

Paradigma der Adressierung, bei dem alle adressierbaren Einheiten dem Programm als einzelner **linearer Adressraum** erscheint

Paradigma der Adressierung, bei dem alle adressierbaren Einheiten dem Programm als einzelner **linearer Adressraum** erscheint

- eindimensionaler Adressraum im Stile von Seitenadressierung
 - sämtliche (*online*) Information im System kann direkt referenzierbar sein¹
 - gebündelt als logische Segmente, erfasst durch logische Deskriptoren

¹Erstmalig realisiert mit Multics [18], aber durch physische Segmentierung.

Paradigma der Adressierung, bei dem alle adressierbaren Einheiten dem Programm als einzelner **linearer Adressraum** erscheint

- eindimensionaler Adressraum im Stile von Seitenadressierung

- jede Entität ist als (seitennummeriertes) Segment ausgeprägt
 - das durch einen/mehrere Seitendeskriptor/en repräsentiert ist
 - für alle Prozesse angelegt in einer geeigneten Tabellenstruktur
 - 5-stufige Seitentabelle, invertierte Seitentabelle, ..., *guarded page table* [15]

¹Erstmalig realisiert mit Multics [18], aber durch physische Segmentierung.

Paradigma der Adressierung, bei dem alle adressierbaren Einheiten dem Programm als einzelner **linearer Adressraum** erscheint

- eindimensionaler Adressraum im Stile von Seitenadressierung
- jede Entität ist als (seitennummeriertes) Segment ausgeprägt
- gemeinsame Nutzung setzt die Weitergabe ihrer Adressen voraus
 - dieselben Adressen selektieren denselben Deskriptor: dieselben Attribute
 - verschiedene Prozesse erhalten darüber auch nur dieselben Zugriffsrechte

¹Erstmalig realisiert mit Multics [18], aber durch physische Segmentierung.

Paradigma der Adressierung, bei dem alle adressierbaren Einheiten dem Programm als einzelner **linearer Adressraum** erscheint

- eindimensionaler Adressraum im Stile von Seitenadressierung

- jede Entität ist als (seitennummeriertes) Segment ausgeprägt

- gemeinsame Nutzung setzt die Weitergabe ihrer Adressen voraus

- Differenzierung von Zugriffsrechten erfordert zusätzliche Maßnahmen
 - i Replikation, Anpassung und Gruppierung von Deskriptoren
 - ii Segmente mit einer **Befähigung** (*capability*, [7]) assoziieren

¹Erstmalig realisiert mit Multics [18], aber durch physische Segmentierung.

Bereiche, die 2^{64} Adressen umfassen, sind so groß, dass die einzelnen darin enthaltenen Entitäten (Objekte) nicht mehr aufzählbar sind²

²Angenommen, ein Zähler schritt dauert 1 ns. Dann muss der Rechner etwa 584,9 Jahre durchgängig im Betrieb sein, um alle 2^{64} Adressen aufzuzählen.

Bereiche, die 2^{64} Adressen umfassen, sind so groß, dass die einzelnen darin enthaltenen Entitäten (Objekte) nicht mehr aufzählbar sind²

- damit sind diese Entitäten bereits sehr gut vor Zugriffen geschützt
 - man muss ihre Adresse schon kennen, sie zu erraten ist hoffnungslos
 - die **Wahrscheinlichkeit**, eine unbekannte Entität zu erreichen, ist gering

²Angenommen, ein Zähler Schritt dauert 1 ns. Dann muss der Rechner etwa 584,9 Jahre durchgängig im Betrieb sein, um alle 2^{64} Adressen aufzuzählen.

Bereiche, die 2^{64} Adressen umfassen, sind so groß, dass die einzelnen darin enthaltenen Entitäten (Objekte) nicht mehr aufzählbar sind²

- damit sind diese Entitäten bereits sehr gut vor Zugriffen geschützt
- dieser Schutz lässt sich durch **Randomisierung** weiter verfeinern
 - die Platzierungsstrategie liefert die Adresse im „winzigen“ Hauptspeicher
 - diese reale Adresse ist einer logischen/virtuellen Adresse zuzuordnen
 - das Betriebssystem nimmt die **Adressabbildung** (*address mapping*) vor
 - bei Seitenadressierung ist der Adresswert ein Vielfaches der Seitengröße
 - die zugeordnete Adresse wäre ein zufälliger Wert im Bereich $[0, 2^n - 1]$
 - mit $n \leq 2^{64-p}$, wobei $p = \log_2(\text{Seitengröße})$
 - sowie $n/\text{Seitengröße} = 0$, also ein seitenausgerichteter Wert

²Angenommen, ein Zähler Schritt dauert 1 ns. Dann muss der Rechner etwa 584,9 Jahre durchgängig im Betrieb sein, um alle 2^{64} Adressen aufzuzählen.

Bereiche, die 2^{64} Adressen umfassen, sind so groß, dass die einzelnen darin enthaltenen Entitäten (Objekte) nicht mehr aufzählbar sind²

- damit sind diese Entitäten bereits sehr gut vor Zugriffen geschützt

- dieser Schutz lässt sich durch **Randomisierung** weiter verfeinern

- die Adresse n muss anderen Prozessen explizit kommuniziert werden, damit sie auf die betreffende Entität zugreifen können

²Angenommen, ein Zähler Schritt dauert 1 ns. Dann muss der Rechner etwa 584,9 Jahre durchgängig im Betrieb sein, um alle 2^{64} Adressen aufzuzählen.

- `mmap(2)` mit der gewöhnlichen (Linux-) Semantik ist ein Problem



- `mmap(2)` mit der gewöhnlichen (Linux-) Semantik ist ein Problem
 - durch Anforderung hinreichend großer Bereiche, lässt sich relativ schnell ein 64-Bit-Raum „abtasten“
 - beispielsweise 4GB große Bereiche
 - dann ist der 64-Bit-Raum mit höchstens 2^{32} Systemaufrufen abgesucht




- `mmap(2)` mit der gewöhnlichen (Linux-) Semantik ist ein Problem
 - durch Anforderung hinreichend großer Bereiche, lässt sich relativ schnell ein 64-Bit-Raum „abtasten“
 - beispielsweise 4GB große Bereiche
 - dann ist der 64-Bit-Raum mit höchstens 2^{32} Systemaufrufen abgesucht ☹
 - bei 1 ms pro 4GB Systemaufruf wäre der 64-Bit-Raum in 50 Tagen geprüft³

³Mit größeren Bereichen wie bspw. 256GB sind es nur noch 9 Stunden, ggf. lässt sich auch `MAP_NORESERVE` geeignet nutzen, um nur noch in Minuten durchzukommen. Andere Tricks zur „Nadelsuche“ sind in [10] beschrieben.

- `mmap(2)` mit der gewöhnlichen (Linux-) Semantik ist ein Problem
 - durch Anforderung hinreichend großer Bereiche, lässt sich relativ schnell ein 64-Bit-Raum „abtasten“
 - beispielsweise 4GB große Bereiche
 - das ließe sich „rekursiv“ im „buddy“-Verfahren mit kleineren Einheiten verfeinern, falls die Erzeugung einer größeren Abbildung scheitert
 - so wäre es möglich, sich auf Seitengranularität an etwas heranzutasten



- `mmap(2)` mit der gewöhnlichen (Linux-) Semantik ist ein Problem
 - durch Anforderung hinreichend großer Bereiche, lässt sich relativ schnell ein 64-Bit-Raum „abtasten“
 - beispielsweise 4GB große Bereiche
 - das ließe sich „rekursiv“ im „buddy“-Verfahren mit kleineren Einheiten verfeinern, falls die Erzeugung einer größeren Abbildung scheitert
 - das Betriebssystem könnte ein reguläres Muster dieses Systemaufrufs sehr ähnlich zu [5] erkennen und Gegenmaßnahmen ergreifen
 - aber ebenso könnte sich der „Angreiferprozess“ darauf einstellen 
-

- Replikation, Anpassung und Gruppierung von **Deskriptoren**
 - für abweichende Zugriffsrechte, eigene Deskriptoren einrichten
 - bis auf Zugriffsrechte gleichen alle anderen Deskriptoreinträge dem Original

- Replikation, Anpassung und Gruppierung von **Deskriptoren**
 - für abweichende Zugriffsrechte, eigene Deskriptoren einrichten und
 - bis auf Zugriffsrechte gleichen alle anderen Deskriptoreinträge dem Original
 - in einer eigenen Deskriptortabelle gruppieren \rightsquigarrow Schutzdomäne einrichten

- Replikation, Anpassung und Gruppierung von **Deskriptoren**
 - für abweichende Zugriffsrechte, eigene Deskriptoren einrichten und
 - bis auf Zugriffsrechte gleichen alle anderen Deskriptoreinträge dem Original
 - in einer eigenen Deskriptortabelle gruppieren \rightsquigarrow Schutzdomäne einrichten
 - ↳ damit werden jedoch nur die typischen „Hardwarezugriffsrechte“ erfasst:
zugreifen, lesen, schreiben, ausführen

- Replikation, Anpassung und Gruppierung von **Deskriptoren**

- Adressen mit einer **Befähigung** (*capability*, [7]) assoziieren
 - allgemein Subjekten Rechte für Operationen auf Objekte zuschreiben [12]

- Replikation, Anpassung und Gruppierung von **Deskriptoren**

- Adressen mit einer **Befähigung** (*capability*, [7]) assoziieren
 - allgemein Subjekten Rechte für Operationen auf Objekte zuschreiben [12]
 - Objekt**– Entität, zu der der Zugriff kontrolliert werden muss
 - Seiten, Segmente, Dateien, Programme, Geräte, Maschinenbefehle

- Replikation, Anpassung und Gruppierung von **Deskriptoren**

- Adressen mit einer **Befähigung** (*capability*, [7]) assoziieren
 - allgemein Subjekten Rechte für Operationen auf Objekte zuschreiben [12]

Subjekt– aktive Entität, deren Zugriff auf Objekte kontrolliert werden muss

- Paar (*Prozess, Domäne*): Schutzdomäne, in der ein Prozess operiert

- Replikation, Anpassung und Gruppierung von **Deskriptoren**

- Adressen mit einer **Befähigung** (*capability*, [7]) assoziieren
 - allgemein Subjekten Rechte für Operationen auf Objekte zuschreiben [12]
 - Objekt**– Entität, zu der der Zugriff kontrolliert werden muss
 - Seiten, Segmente, Dateien, Programme, Geräte, Maschinenbefehle
 - Subjekt**– aktive Entität, deren Zugriff auf Objekte kontrolliert werden muss
 - Paar (*Prozess, Domäne*): Schutzdomäne, in der ein Prozess operiert
 - sicherstellen, dass die Rechtezuordnung nicht gefälscht werden kann !!!

- Replikation, Anpassung und Gruppierung von **Deskriptoren**

- Adressen mit einer **Befähigung** (*capability*, [7]) assoziieren

↪ damit auch weitergehende, nicht nur einfache Zugriffe erfassende Rechte einräumen: übertragen, bewilligen, löschen, erzeugen, zerstören

Subjekte verfügen allein durch den Besitz einer Befähigung über die darüber definierten Rechte beim Objektzugriff

- Befähigungen sind daher selbst zu schützen

Subjekte verfügen allein durch den Besitz einer Befähigung über die darüber definierten Rechte beim Objektzugriff

- Befähigungen sind daher selbst zu schützen, beispielsweise:
 - i als Liste (*capability list, C-list*, [7]) gespeichert in speziellen Segmenten, überwacht vom Betriebssystem
 - ii durch ein zusätzliches Kennwort [1], vergeben vom Objektverwalter bei Zuordnung der Adresse (des Deskriptors) zu einem Objekt

Subjekte verfügen allein durch den Besitz einer Befähigung über die darüber definierten Rechte beim Objektzugriff

- Befähigungen sind daher selbst zu schützen

- für SASOS attraktiv ist die kennwortgeschützte Befähigung [1]
 - sie ist frei kopier-, speicher- und kommunizierbar, wie andere Daten auch
 - für ihre Verwendung muss das Betriebssystem nicht einbezogen werden

Subjekte verfügen allein durch den Besitz einer Befähigung über die darüber definierten Rechte beim Objektzugriff

- Befähigungen sind daher selbst zu schützen

- für SASOS attraktiv ist die kennwortgeschützte Befähigung [1]

- eine solche „spärliche“ Befähigung C versteht sich als Tupel (A, P)
 - mit dem Namen des Objektes bzw. dessen Adresse $A = [0, 2^{64} - 1]$
 - sowie dem Kennwort P , eine wenigstens 64-Bit große Zufallszahl
 - die ein unberechenbarer physikalischer Prozess [23] generiert
 - z.B. thermisches Rauschen [1], durch einen A/D-Wandler dargestellt

Einadressraumsystem

Einleitung

Rückblick

Modell

Adressierung

Schutz

Gedankenspiel

Diskurs

Zusammenfassung

- für isolierte Anwendungen ist SAS ein Implementierungsdetail⁴
 - keinen Kontakt/Austausch mit Prozessen anderer Anwendungsprogramme
 - keinen bewussten Kontakt/Austausch mit dem Betriebssystem

⁴Soweit es die funktionalen Eigenschaften betrifft.

- für isolierte Anwendungen ist SAS ein Implementierungsdetail⁴
 - keinen Kontakt/Austausch mit Prozessen anderer Anwendungsprogramme
 - keinen bewussten Kontakt/Austausch mit dem Betriebssystem
 - ↳ eigentlich gibt es nichts, was mit MAS nicht ebenso ginge
 - ↳ andersherum wäre SAS diesen Anwendungen auch kein Hindernis

⁴Soweit es die funktionalen Eigenschaften betrifft.

- für isolierte Anwendungen ist SAS ein Implementierungsdetail⁴

- für **kooperative Anwendungen** kann SAS von Vorteil sein
 - effiziente/direkte Kommunikation durch simplen Zeigeraustausch
 - *data shipping*
 - *function shipping*
 - sowohl in horizontaler als auch vertikaler Hinsicht
 - innerhalb der Anwendungsprogrammebene zwischen Prozessexemplaren
 - ebenenübergreifend zwischen Anwendungsprogramm- und Betriebssystemebene

⁴Soweit es die funktionalen Eigenschaften betrifft.

- für isolierte Anwendungen ist SAS ein Implementierungsdetail⁴

- für **kooperative Anwendungen** kann SAS von Vorteil sein

- aus Betriebssystemensicht ermöglicht SAS „leichtgewichtiger“ Dienste
 - die mit besseren nichtfunktionalen Eigenschaften behaftet sind

⁴Soweit es die funktionalen Eigenschaften betrifft.

- Anwendungen in eine **speichersichere Zwischensprache** kompilieren
 - in einen geeigneten Bytecode (Zwischenkode) für eine virtuelle Maschine
 - bedarfssynchrone (*just in time*, JIT) Übersetzung dann im Betriebssystem
 - das die übersetzten Programmbestandteile in einem Zwischenspeicher hält
 - das Betriebssystem als Ganzes würde durch diesen Vorgang „urgeladen“
 - bis auf spezielle Module, die in Assembler von Hand zu schreiben sind⁵

⁵Wenn es an einer Systemimplementierungssprache mangelt (vgl. [22]).

- Anwendungen in eine **speichersichere Zwischensprache** kompilieren
 - in einen geeigneten Bytecode (Zwischenkode) für eine virtuelle Maschine

- alle Programme könnten derselben Schutzdomäne zugeteilt sein
 - inkl. Betriebssystemprogramme wäre das der **privilegierte Arbeitsmodus**
 - also der Modus, in dem die CPU initial – beim Anschalten – läuft

- Anwendungen in eine **speichersichere Zwischensprache** kompilieren
 - in einen geeigneten Bytecode (Zwischenkode) für eine virtuelle Maschine

 - alle Programme könnten derselben Schutzdomäne zugeteilt sein
 - inkl. Betriebssystemprogramme wäre das der **privilegierte Arbeitsmodus**
 - also der Modus, in dem die CPU initial – beim Anschalten – läuft
 - dann müsste aber auch das Betriebssystem typsicher programmiert sein
 - ansonsten gehen „Systemaufrufe“ an typunsichere externe Unterprogramme
 - diese könnten beliebige Unterprogramme sein, nicht nur Systemprogramme
-

- Anwendungen in eine **speichersichere Zwischensprache** kompilieren
 - in einen geeigneten Bytecode (Zwischenkode) für eine virtuelle Maschine

 - alle Programme könnten derselben Schutzdomäne zugeteilt sein

 - die semantische äquivalente **virtuelle Maschine** wäre zwingend
 - direktes Absetzen privilegierter Maschinenbefehle muss unmöglich sein
 - direktes Durchgreifen auf die (reale) Befehlssatzebene ist verboten↪ obligatorisch für alle Anwendungsprogramme
-

- verschiedene „virtuelle Maschinen“ mit verschiedenen Fähigkeiten
 - Prozesse der **Anwendungsprogrammebene** sind nicht privilegiert
 - Prozesse der **Betriebssystemebene** sind privilegiert

- verschiedene „virtuelle Maschinen“ mit verschiedenen Fähigkeiten
 - Prozesse der **Anwendungsprogrammebene** sind nicht privilegiert
 - Prozesse der **Betriebssystemebene** sind privilegiert

Zweiklassengesellschaft

Mit den Anwendungsprogrammprozessen als die „Mittellosen“ und den Betriebssystemprozessen als die „Wohlhabenden“.

↪ gewöhnliche (klassische) zweistufige Qualifizierung der Prozesse genügt

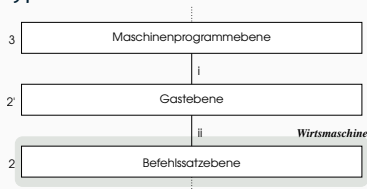
- verschiedene „virtuelle Maschinen“ mit verschiedenen Fähigkeiten
 - Prozesse der **Anwendungsprogrammebene** sind nicht privilegiert
 - Prozesse der **Betriebssystemebene** sind privilegiert

- zum Absetzen privilegierter Befehle muss ein Prozess befähigt sein
 - dies definiert die Schutzdomäne, aus der heraus er den Befehl absetzt
 - unprivilegiert**– die CPU „trapped“ den Prozess, das OS prüft seine Rechte
 - ↪ Prozesse müssen Rechte in Vertrauen (des OS) erhalten
 - privilegiert**– die CPU lässt den Prozess gewähren

- verschiedene „virtuelle Maschinen“ mit verschiedenen Fähigkeiten
 - Prozesse der **Anwendungsprogrammebene** sind nicht privilegiert
 - Prozesse der **Betriebssystemebene** sind privilegiert

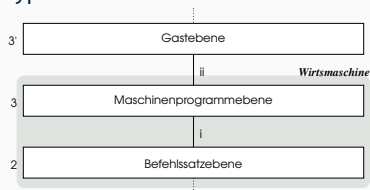
- zum Absetzen privilegierter Befehle muss ein Prozess befähigt sein
 - **Teilinterpretation** [14] privilegierter Befehle unprivilegierter Prozesse
 - durch einen speziellen **Hypervisor**, Monitor für virtuelle Maschinen
 - durch einen Exokern [9], falls mehr als nur diese Befehle zu beachten sind

■ Typ I VMM



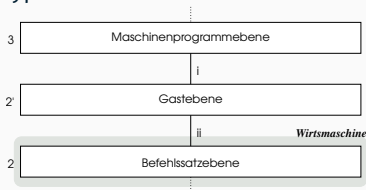
- läuft auf einer „nackten“ Wirtmaschine
- unter keinem Betriebssystem

■ Typ II VMM

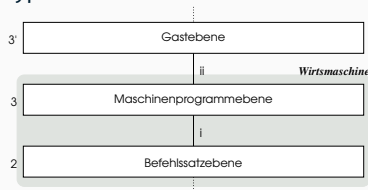


- läuft auf einer erweiterten Wirtmaschine
- unter dem Wirtsbetriebssystem

■ Typ I VMM

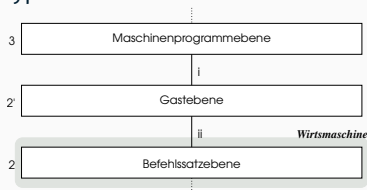


■ Typ II VMM



- beiden gemeinsames Operationsprinzip ist die **Teilinterpretation**:
 - i durch das Betriebssystem (Typ I) bzw. Wirtbetriebssystem (Typ II)
 - ii durch den VMM

■ Typ I VMM

■ Gegenstand der Teilinterpretation sind **sensitive Befehle**

- jeder Befehl, dessen direkte Ausführung durch die VM nicht tolerierbar ist
 - privilegierte Befehle ausgeführt im unprivilegierten Modus \rightsquigarrow Trap
 - bei Virtualisierung auch unprivilegierte Befehle mit kritischen Seiteneffekten



- typische Anforderungen an die Befehlssatzebene [11, S. 47–53]:
 1. annähernd äquivalente Ausführung der meisten unprivilegierten Befehle im System- und Anwendungsmodus des Rechnersystems
 2. Schutz von Programmen, die im Systemmodus ausgeführt werden
 3. Abfangvorrichtung („Falle“, *trap*) für **sensitive Befehle**:
 - a Änderung/Abfrage des Systemzustands (z.B. Arbeitsmodus des Rechners)
 - b Änderung/Abfrage des Zustands reservierter Register oder Speicherstellen
 - c Referenzierung des (für 2. erforderlichen) Schutzsystems
 - d Ein-/Ausgabe
-

2. Schutz von Programmen, die im Systemmodus ausgeführt werden
 3. Abfangvorrichtung („Falle“, *trap*) für **sensitive Befehle**:
 - a Änderung/Abfrage des Systemzustands (z.B. Arbeitsmodus des Rechners)
 - b Änderung/Abfrage des Zustands reservierter Register oder Speicherstellen
- relevant für SAS sind Mechanismen der CPU für 2. und 3.
 - bei 3.a und 3.b jedoch nur Schutz gegen Änderungsversuche
 - nur die **partielle Virtualisierung** der CPU ist erforderlich⁶

⁶So sind unprivilegierte sensitive Befehle (s. [19] für Pentium) unkritisch.

2. Schutz von Programmen, die im Systemmodus ausgeführt werden
3. Abfangvorrichtung („Falle“, *trap*) für **sensitive Befehle**:

- relevant für SAS sind Mechanismen der CPU für 2. und 3.
 - herkömmliche Differenzierung des Arbeitsmodus' in *user/system mode*
 - *cli/sti, in/out, ..., mov* mit Operand *%cr3* sind im *user mode* tabu
 - beim *trap* wird ein berechtigter Prozess in den *system mode* befördert und wechselt damit gegebenenfalls seine Schutzdomäne
 - als Systemaufruf verpackt impliziert die Rückkehr in die alte Schutzdomäne
-

- Erzeugung eines Prozessexemplars, indem der elterliche Adressraum dupliziert wird, ist kontraproduktiv zu SAS

- Erzeugung eines Prozessexemplars, indem der elterliche Adressraum dupliziert wird, ist kontraproduktiv zu SAS
 - fork(2) impliziert die **Mehrdeutigkeit** von Adressen
 - auch dann, wenn die dahinter stehenden Informationen nicht kopiert werden
 - egal, ob diesem Systemaufruf im Programm direkt ein exec(2) folgt
 - eine solche Annahme darf das Betriebssystem nicht treffen, um zu optimieren
 - denn dann würde es das Anwendungsprogramm „benutzen“ [21]

- Erzeugung eines Prozessexemplars, indem der elterliche Adressraum dupliziert wird, ist kontraproduktiv zu SAS

- um Anwendungsprogramme für ein **Altsystem** (*legacy system/code*) zu unterstützen, sind **Behelfslösungen** adäquat
 - das Textsegment wäre kein Problem, kann gemeinsam genutzt werden
 - ansonsten positionsunabhängiger Code
 - problematisch sind Daten- und Stapelsegment
 - positionsunabhängige Zugriffe, relativ über ein Basisregister
 - statische Programmanalyse, um Zeiger zu lokalisieren – im Zusammenspiel mit einem Laufzeitsystem, um Zeiger zu relozieren (à la *garbage collection*)

- Erzeugung eines Prozessexemplars, indem der elterliche Adressraum dupliziert wird, ist kontraproduktiv zu SAS

- um Anwendungsprogramme für ein **Altsystem** (*legacy system/code*) zu unterstützen, sind **Behelfslösungen** adäquat
 - das Textsegment wäre kein Problem, kann gemeinsam genutzt werden
 - ansonsten positionsunabhängiger Code ~> **Quelltext**
 - problematisch sind Daten- und Stapelsegment ~> **Quelltext**
 - positionsunabhängige Zugriffe, relativ über ein Basisregister
 - statische Programmanalyse, um Zeiger zu lokalisieren – im Zusammenspiel mit einem Laufzeitsystem, um Zeiger zu relozieren (à la *garbage collection*)

- Erzeugung eines Prozessexemplars, indem der elterliche Adressraum dupliziert wird, ist kontraproduktiv zu SAS

- um Anwendungsprogramme für ein **Altsystem** (*legacy system/code*) zu unterstützen, sind **Behelfslösungen** adäquat

- SAS ist **disruptiv**, soweit es Standardschnittstellen betrifft, die den Stand der Technik bei Betriebssystemen (POSIX) definieren

Einadressraumsystem

Einleitung

Rückblick

Modell

Adressierung

Schutz

Gedankenspiel

Diskurs

Zusammenfassung

■ Einadressraumsysteme

- sehen die Belange „Adressierung“ und „Schutz“ getrennt voneinander
- profitieren von Prozessoren mit extrabreiten Adressen $A = [0, 2^{64} - 1]$
- gebrauchen spärliche, kennwortgeschützte Befähigungen $C = (A, P)$
- „benutzen“ ansonsten herkömmliche Adressraumverwaltungshardware

■ Einadressraumsysteme

- sehen die Belange „Adressierung“ und „Schutz“ getrennt voneinander
- profitieren von Prozessoren mit extrabreiten Adressen $A = [0, 2^{64} - 1]$
- gebrauchen spärliche, kennwortgeschützte Befähigungen $C = (A, P)$
- „benutzen“ ansonsten herkömmliche Adressraumverwaltungshardware

In essence, protection in a SASOS is provided not by controlling what is in the address space, but by controlling which parts of it can be accessed. [13, S. 7]

■ Einadressraumsysteme

- sehen die Belange „Adressierung“ und „Schutz“ getrennt voneinander
- profitieren von Prozessoren mit extrabreiten Adressen $A = [0, 2^{64} - 1]$
- gebrauchen spärliche, kennwortgeschützte Befähigungen $C = (A, P)$
- „benutzen“ ansonsten herkömmliche Adressraumverwaltungshardware

In essence, protection in a SASOS is provided not by controlling what is in the address space, but by controlling which parts of it can be accessed. [13, S. 7]

- **Abwärtskompatibilität** zu bewahren, ist eine große Herausforderung
 - SAS ist **disruptiv**, soweit es Standardschnittstellen betrifft, die den Stand der Technik bei Betriebssystemen (POSIX) definieren

■ Einadressraumsysteme

- sehen die Belange „Adressierung“ und „Schutz“ getrennt voneinander
- profitieren von Prozessoren mit extrabreiten Adressen $A = [0, 2^{64} - 1]$
- gebrauchen spärliche, kennwortgeschützte Befähigungen $C = (A, P)$
- „benutzen“ ansonsten herkömmliche Adressraumverwaltungshardware

In essence, protection in a SASOS is provided not by controlling what is in the address space, but by controlling which parts of it can be accessed. [13, S. 7]

- **Abwärtskompatibilität** zu bewahren, ist eine große Herausforderung
 - SAS ist **disruptiv**, soweit es Standardschnittstellen betrifft, die den Stand der Technik bei Betriebssystemen (POSIX) definieren
- die RISC-V Spezifikation sieht bereits eine 128-Bit Version vor...

- [1] ANDERSON, M. ; POSE, R. D. ; WALLACE, C. S.:
A Password-Capability System.
In: *The Computer Journal* 29 (1986), Nr. 1, S. 1–8
- [2] CHASE, J. ; LEVY, H. ; BAKER-HARVEY, M. ; LAZOWSKA, E. :
Opal: a single address space system for 64-bit architecture address space.
In: *Proceedings Third Workshop on Workstation Operating Systems (WWOS 1992)*, 1992, S. 80–85
- [3] CHASE, J. S. ; LEVY, H. M. ; FREELEY, M. J. ; LAZOWSKA, E. D.:
Sharing and Protection in a Single-Address-Space Operating System.
In: *ACM Transactions on Computer Systems* 12 (1994), Nov., Nr. 4, S. 271–307

- [4] CORBATÓ, F. J. ; DAGGETT, M. M. ; DALEY, R. C.:
An Experimental Time-Sharing System.
In: *Proceedings of the 1962 Spring Joint Computer Conference (AFIPS '62)*
American Federation of Information Processing Societies, AFIPS Press,
1962, S. 335–344
- [5] CUCINOTTA, T. ; CHECCONI, F. ; ABENI, L. ; PALOPOLI, L. :
Self-Tuning Schedulers for Legacy Real-Time Applications.
In: *Proceedings of the 5th European Conference on Computer Systems*
(EuroSys '10), Association for Computing Machinery, 2010, S. 55–68
- [6] DALEY, R. C. ; DENNIS, J. B.:
Virtual Memory, Processes, and Sharing in MULTICS.
In: *Communications of the ACM* 11 (1968), Mai, Nr. 5, S. 306–312
- [7] DENNIS, J. B. ; HORN, E. C. V.:
Programming Semantics for Multiprogrammed Computations.
In: *Communications of the ACM* 9 (1966), März, Nr. 3, S. 143–155

- [8] DIJKSTRA, E. W.:
On the Role of Scientific Thought.
<http://www.cs.utexas.edu/users/EWD/ewd04xx/EWD447.PDF>,
Aug. 1974
- [9] ENGLER, D. R. ; KAASHOEK, M. F. ; O'TOOLE, J. :
Exokernel: An Operating System Architecture for Application-Level Resource Management.
In: JONES, M. B. (Hrsg.): *Proceedings of the 15th ACM Symposium on Operating System Principles (SOSP '95)*, ACM Press, 1995. –
ISBN 0-89791-715-4, S. 251-266
- [10] GISBERT, H. M. ; RIPOLI, I. :
On the Effectiveness of Full-ASLR on 64-bit Linux.
In: *In-depth Security Conference 2014 Europe (DeepSec 2014)* DeepSec GmbH, 2014, S. 1-9

- [11] GOLDBERG, R. P.:
Architectural Principles for Virtual Computer Systems / Harvard University, Electronic Systems Division.
Cambridge, MA, USA, Febr. 1973 (ESD-TR-73-105). –
PhD Thesis
- [12] GRAHAM, G. S. ; DENNING, P. J.:
Protection—Principles and Practice.
In: *Proceedings of the Spring Joint Computer Conference (AFIPS '72)*.
New York, NY, USA : ACM, 1972, S. 417–429
- [13] HEISER, G. ; ELPHINSTONE, K. ; VOCHTELOO, J. ; RUSSEL, S. ; LIEDTKE, J. :
The Mungi Single-Address-Space Operating System.
In: *Software—Practice and Experience* 18 (1998), Jul., Nr. 9
- [14] KLEINÖDER, J. ; SCHRÖDER-PREIKSCHAT, W. :
Betriebssystemmaschine.
In: LEHRSTUHL INFORMATIK 4 (Hrsg.): *Systemprogrammierung*.
FAU Erlangen-Nürnberg, 2015 (Vorlesungsfolien), Kapitel 5.3

- [15] LIEDTKE, J. :
On the Realization of Huge Sparsely-Occupied and Fine-Grained Address Spaces, TU Berlin, Diss., 1996
- [16] NELSON, B. J.:
Remote Procedure Call.
Pittsburg, PA, USA, Department of Computer Science, Carnegie-Mellon University, Diss., Mai 1981
- [17] NELSON, R. A.:
Mapping Devices and the M44 Data Processing System / IBM Research Division.
IBM Watson Research Center, Yorktown Heights, New York, Okt. 1964 (RC1303). –
Research Report
- [18] ORGANICK, E. I.:
The Multics System: An Examination of its Structure.
MIT Press, 1972. –
ISBN 0-262-15012-3

- [19] ROBIN, J. S. ; IRVINE, C. E.:
Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor.
In: *Proceedings of 9th USENIX Security Symposium (SSYM'00)*, USENIX Association, 2000, S. 1–16
- [20] SCHRÖDER-PREIKSCHAT, W. ; LEHRSTUHL INFORMATIK 4 (Hrsg.):
Betriebssystemtechnik – Adressräume: Trennung, Zugriff, Schutz.
FAU Erlangen-Nürnberg, 2013 (Vorlesungsfolien)
- [21] SCHRÖDER-PREIKSCHAT, W. :
Hierarchien.
In: [20], Kapitel 4
- [22] SCHRÖDER-PREIKSCHAT, W. :
Sprachbasierte Systeme.
In: [20], Kapitel 7

[23] WALLACE, C. S.:

Physically Random Generator.

In: *Computer Systems Science and Engineering* 5 (1990), Apr., Nr. 2, S. 82–88

[24] YARVIN, C. ; BUKOWSKI, R. ; ANDERSON, T. :

Anonymous RPC: Low-Latency Protection in a 64-Bit Address Space.

In: *USENIX Summer 1993 Technical Conference (USENIX Summer 1993 Technical Conference)* Bd. 1.
Cincinnati, OH : USENIX Association, Jun. 1993, S. 13:1–12