Betriebssystemtechnik

Adressräume: Trennung, Zugriff, Schutz

VI. Adressraummodelle

Wolfgang Schröder-Preikschat

19. Mai 2015

Gliederung

Räumliche Isolation oder Mitbenutzung

- Mehradressraumkonzept
  - Betriebssystem und Maschinenprogramme erhalten private Adressräume
    - die gleichsam Schutzdomänen manifestieren
  - Grundlage dafür ist die Mehrfachnutzung desselben realen Adressbereichs
    - für einen N-Bit Rechner gilt für diesen Adressbereich $A = [0, 2^N - 1]$
  - Adressraum als Entität, der eine gewisse Virtualität zugeschrieben wurde
    - d.h., die nicht physisch, aber in ihrer Funktionalität/Wirkung vorhanden ist
  - Technik dahinter ist die Vervielfachung eines (realen) Adressbereichs:
    i A komplett *oder*
    ii der obere oder untere Teilbereich von $A \rightsquigarrow \text{OOStuBS}_{BST}$
- Einadressraumkonzept
  - Betriebssystem und Maschinenprogramme 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

Gliederung

http://de.wikipedia.org/wiki/Virtualität

Die Eigenschaft einer Sache, nicht in der Form zu existieren, in der sie zu existieren scheint, aber in ihrem Wesen oder ihrer Wirkung einer in dieser Form existierenden Sache zu gleichen.

- Virtualisierung des realen Adressbereichs – nicht des Hauptspeichers!
  - i Vervielfachung von $A = [0, 2^N - 1]$
    - komplett A für Betriebssystem und allen Maschinenprogrammen, jeweils
  - ii Einrichtung von $A_t = [0, 2^N - 1]$, Vervielfachung von $A_p \subset A_t$
    - komplett A_t (total) für das Betriebssystem
    - komplett A_p (partiell) für alle Maschinenprogramme, jeweils
- reale Adressen dieser Bereiche sind nicht wirklich (physisch), wohl aber in ihrer Funktionalität vorhanden
  - hinter jeder dieser Adresse steht eine speicherabbildbare Entität
  - sie referenzieren Entitäten der Programmtexte (d.h., Befehle) oder -daten

↪ Segmentierung und/oder Seitennummerierung von Adressräumen

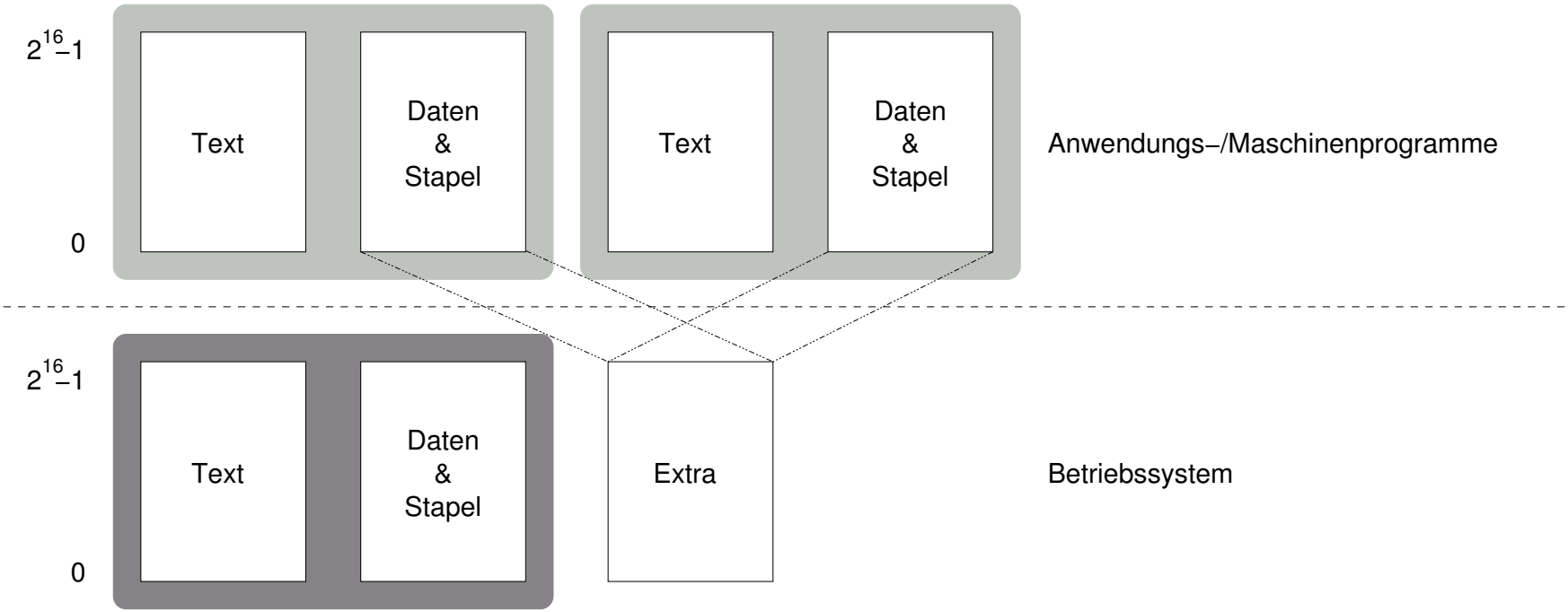
Private Addressräume

Illusion von einem eigenen physischen Adressraum für Betriebssystem und Maschinenprogramme $\rightsquigarrow$ Exklusion

- Vervielfachung des Adressbereichs $A = [0, 2^N - 1]$
  - wobei N bestimmt ist durch die reale Adressbreite des Prozessors
  - evolutionär betrachtet galt/gilt z.B. für $N = 16, 20, 24, 31, 32, 48, 64$
- die MMU[1] verhindert ein Ausbrechen von Prozessen aus A
  - dies gilt für alle durch das Betriebssystem verwalteten Prozesse, *also*
  - sowohl für Maschinenprogramme als auch für das Betriebssystem selbst
- Informationstausch zwischen Programmen mittels Maschinenbefehle
  - des Betriebssystems für die Maschinenprogramme *und* $\mapsto$ Ebene 3
    - Systemaufrufe zur Interprozesskommunikation oder Adressbereichsabbildung
  - der Zentraleinheit für die Betriebssystemprogramme $\mapsto$ Ebene 2
    - privilegierte Befehle zum Lese-/Schreibzugriff auf den Benutzeradressraum
- im Vordergrund steht die strikte Isolation von ganzen Adressräumen

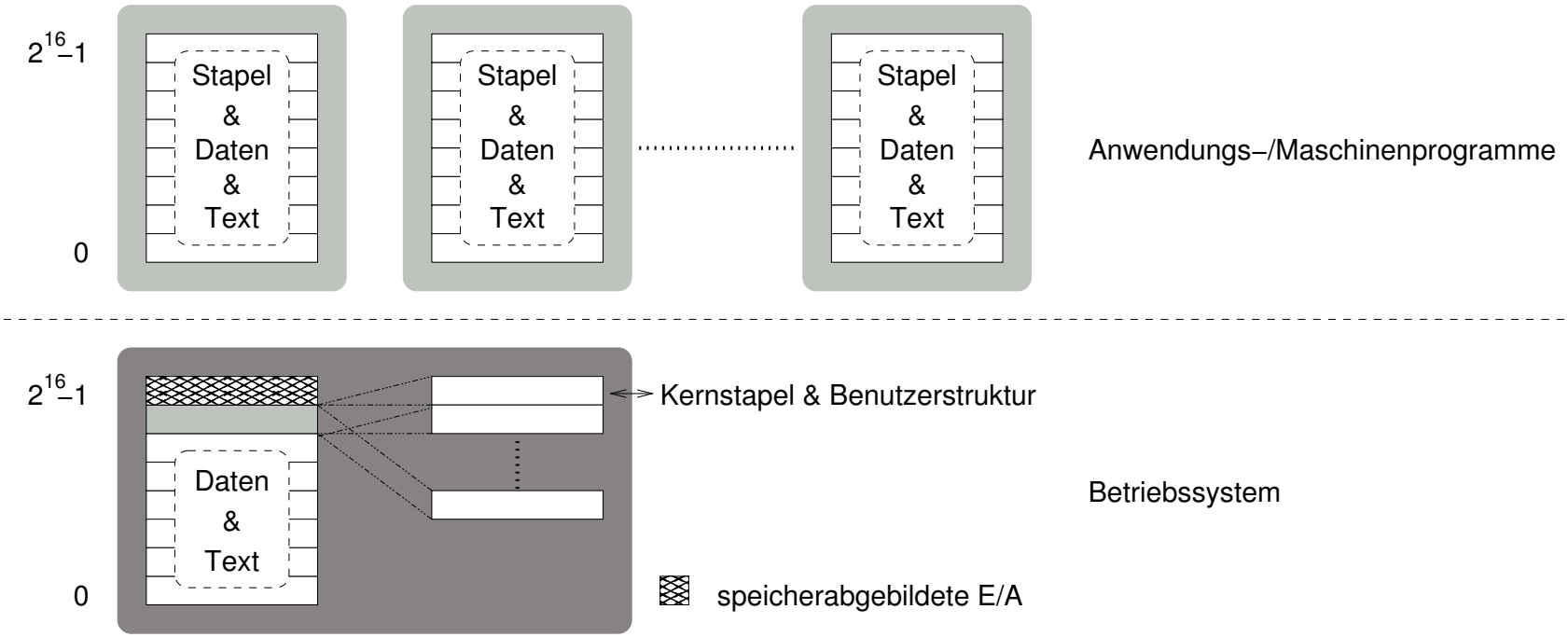
[1] Ebenso eine MPU (*memory protection unit*).

Fensterbasierter Ansatz I



- horizontaler und vertikaler Informationsaustausch
  - horizontal: Interprozesskommunikation (Nachrichtenversenden)
  - vertikal: Zugriffe auf den Benutzeradressraum mittels Extrasegment
- Beispiele von (mikrokernbasierten) Betriebssystemen der Art:
  - QNX [14]: ereignisbasiert, vgl. auch [9]
  - AX [15]: ereignis-/prozedurbasiert, QNX-kompatibel

Spezialbefehlbasierter Ansatz



- horizontaler und vertikaler Informationsaustausch
  - horizontal
    - Interprozesskommunikation (Nachrichtenversenden, *pipe*)
    - seitenbasierte Mitbenutzung in Inkrementen von 64 Bytes
  - vertikal
    - Zugriffe auf den Benutzeradressraum mittels Spezialbefehle
- prominentes Beispiel eines (monolithischen) Betriebssystems der Art:
  - UNIX
    - Version 6 [11, 10], prozedurbasiert

Vertikaler Benutzeradressraumzugriff I

- der Kern muss die Spezialbefehle kontrolliert zur Ausführung bringen
  - `mfpi` *ea*
    - *move from previous instruction space*
    - das von *ea* gelesene Datum wird auf den Kernstapel abgelegt
  - `mtpi` *ea*
    - *move to previous instruction space*
    - das an *ea* zu schreibende Datum wird vom Kernstapel geholt
  - ↪ *ea* ist *effektive Adresse* im Adressraum vor Kernaktivierung
- ungültige Benutzeradressen können diese Befehle scheitern lassen !!!
  - der Kern muss daraufhin seinerseits sein mögliches Scheitern abwenden
- im MMU-Fall wird die Prozessorbetriebsart verfolgt (vgl. [5, S. 2-4]):
  - 00 *kernel mode* ⇝ privilegiert, sicher, vertrauenswürdig
  - 11 *user mode* ⇝ unprivilegiert, unsicher, zweifelhaft
- dafür vorgesehene Bitfelder im Prozessorstatuswort:
  - 15–14 *current mode*
  - 13–12 *previous mode*, vor der letzten Unterbrechung (*trap*, *interrupt*)
- je Betriebsart 8 Seitenadressregister, Umschaltung bei Moduswechsel

Kontrollierte Spezialbefehlausführung

Vorbeugungsmaßnahmen, um ein mögliches Scheitern des Kerns – und damit einen Systemabsturz (*crash*) – abzuwenden

- sind betriebssystemabhängig & gehen optimistisch/pessimistisch vor

  optimistisch
  - Annahme: *ea* ist eine eher gültige Benutzeradresse
  - den Kern auf mögliche Unterbrechung (*trap*) vorbereiten
  - im Ausnahmefall die Ausführung des Kerns fortsetzen

  pessimistisch
  - Annahme: *ea* ist eine eher ungültige Benutzeradresse
  - diese vor Verwendung mit dem Spezialbefehl überprüfen
  - den Ausnahmefall im Kern nicht auftreten lassen

- im Falle eines voll verdrängbaren Kerns offenbart der pessimistische Ansatz eine Laufgefahr (*race hazard*)
  - angenommen, die Prüfung ergab die Gültigkeit der Benutzeradresse *ea*
  - vor Verwendung von *ea* im Kern wird der Prozess aber verdrängt
  - asynchron dazu verändert sich die Adressraumbelegung des Prozesses
  - so dass Benutzeradresse *ea* bei Prozesswiederaufnahme ungültig ist

↪ **beachte**: ein Betriebssystem darf Benutzerprogr. nicht „benutzen“

Optimistisches Vorgehen: Zugriffsoperationen

```
1  gword:                          ; note: r1 holds read address
2    mov   PS, -(sp)               ; save processor status word
3    bis   $340, PS                ; disable interrupts
4    mov   nofault, -(sp)          ; save kernel detour pointer
5    mov   $err, nofault           ; setup detour for trap handler
6    mfpi  (r1)                    ; transfer data onto kernel stack
7    mov   (sp)+, r0               ; receive data just read
8    br    1f                      ; prepare for return...
9  pword:                          ; note: r1 holds write address
10   mov   PS, -(sp)               ; save processor status word
11   bis   $340, PS                ; disable interrupts
12   mov   nofault, -(sp)          ; save kernel detour pointer
13   mov   $err, nofault           ; setup detour for trap handler
14   mov   r0, -(sp)               ; send data to be written
15   mtpi  (r1)                    ; transfer data from kernel stack
16 1:
17   mov   (sp)+, nofault          ; restore kernel detour pointer
18   mov   (sp)+, PS               ; restore processor status word
19   rts   pc
```

Optimistisches Vorgehen: Ausnahme

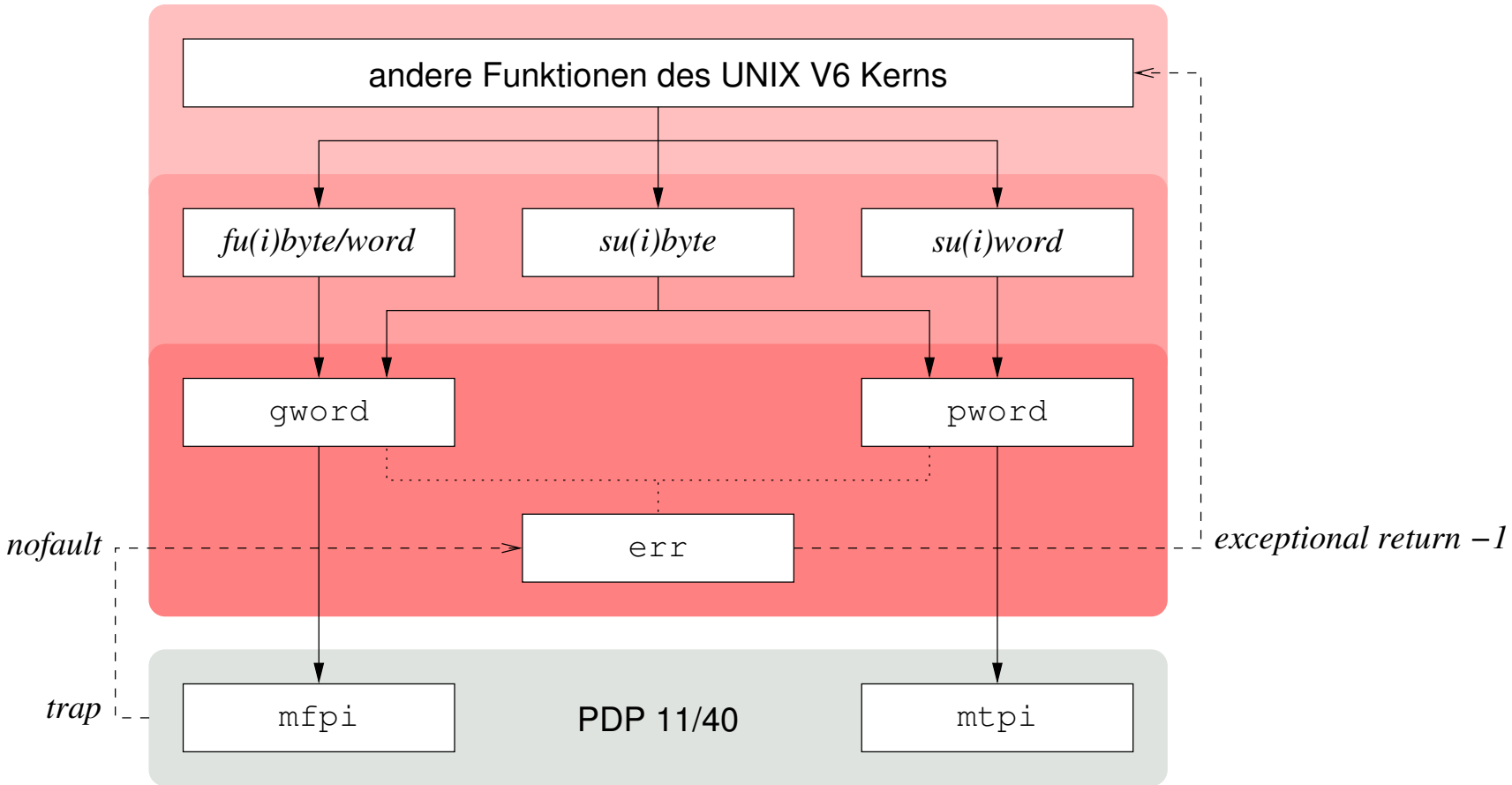
- Abfangen der Ausnahmesituation (*trap*): `mfpi`/`mtpi` ist gescheitert

```
1  trap:
2    mov   PS, -4(sp)       ; magic: needed for code 1f
3    tst   nofault          ; check for kernel detour
4    beq   1f               ; none, continue
5    mov   $1, SSR0         ; else, reinitialize MMU
6    mov   nofault, (sp)    ; overwrite return address and
7    rtt                    ; return from trap
8  1:                       ; fall into normal trap...
```

- „*escape exception*“ [6] für `gword`/`pword`-rufende Programme

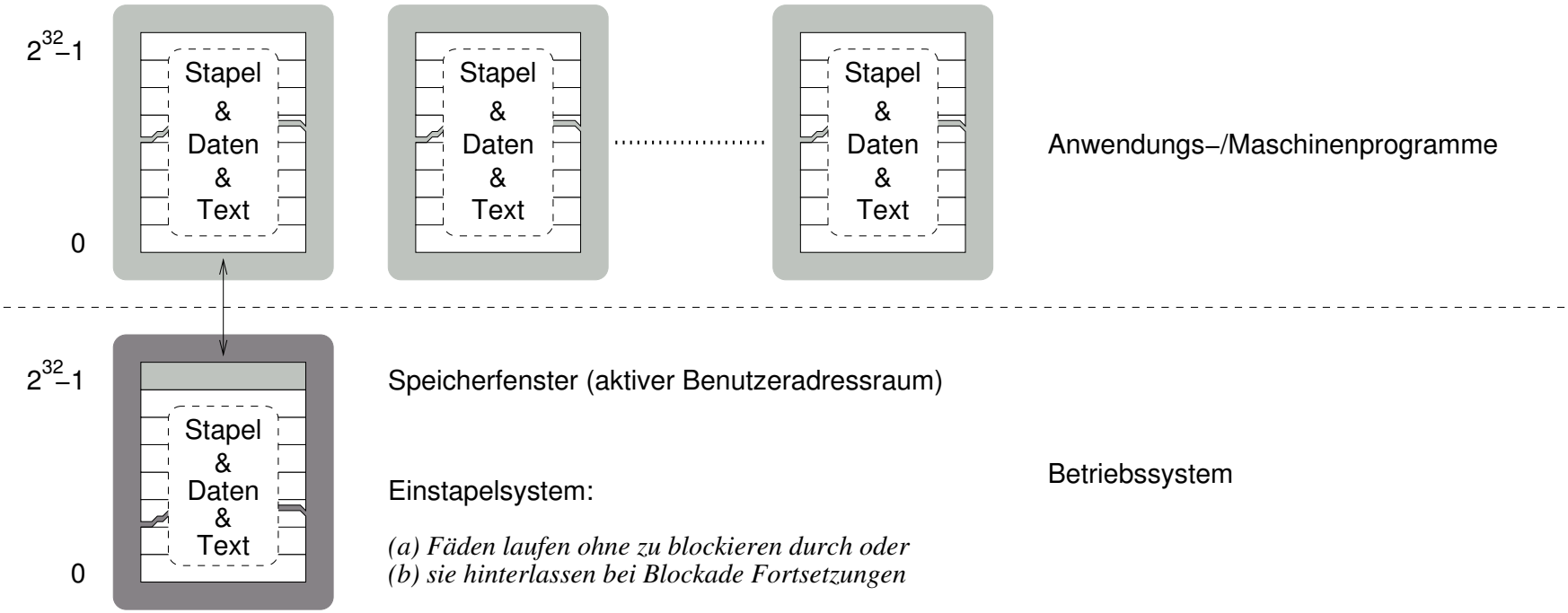
```
1  err:
2    mov   (sp)+, nofault   ; restore kernel detour pointer
3    mov   (sp)+, PS        ; restore processor status word
4    tst   (sp)+            ; forget g/pword return address
5    mov   $-1, r0          ; indicate failure of operation
6    rts   pc               ; return from caller of g/pword
```

Vertikaler Benutzeradressraumzugriff II



- im Ausnahmefall werden die *fetch*/*store*-Operationen abgebrochen
  - sie bilden zusammen mit `gword`, `pword` und `err` ein Modul [13]
  - denn: alle teilen sich dasselbe Wissen über Stapelaufbau/Ereignisabfolge

Fensterbasierter Ansatz II



- horizontaler und vertikaler Informationsaustausch
  - horizontal
    - Interprozesskommunikation (Nachrichtenversenden)
    - seitenbasierte Mitbenutzung
  - vertikal
    - Zugriffe auf den Benutzeradressraum mittels Speicherfenster
- prominentes Beispiel eines Betriebssystems der Art:
  - OS X
    - Hybridkernansatz, ereignis-/prozedurbasiert, 512 MiB Fenster

Partiell private Addressräume

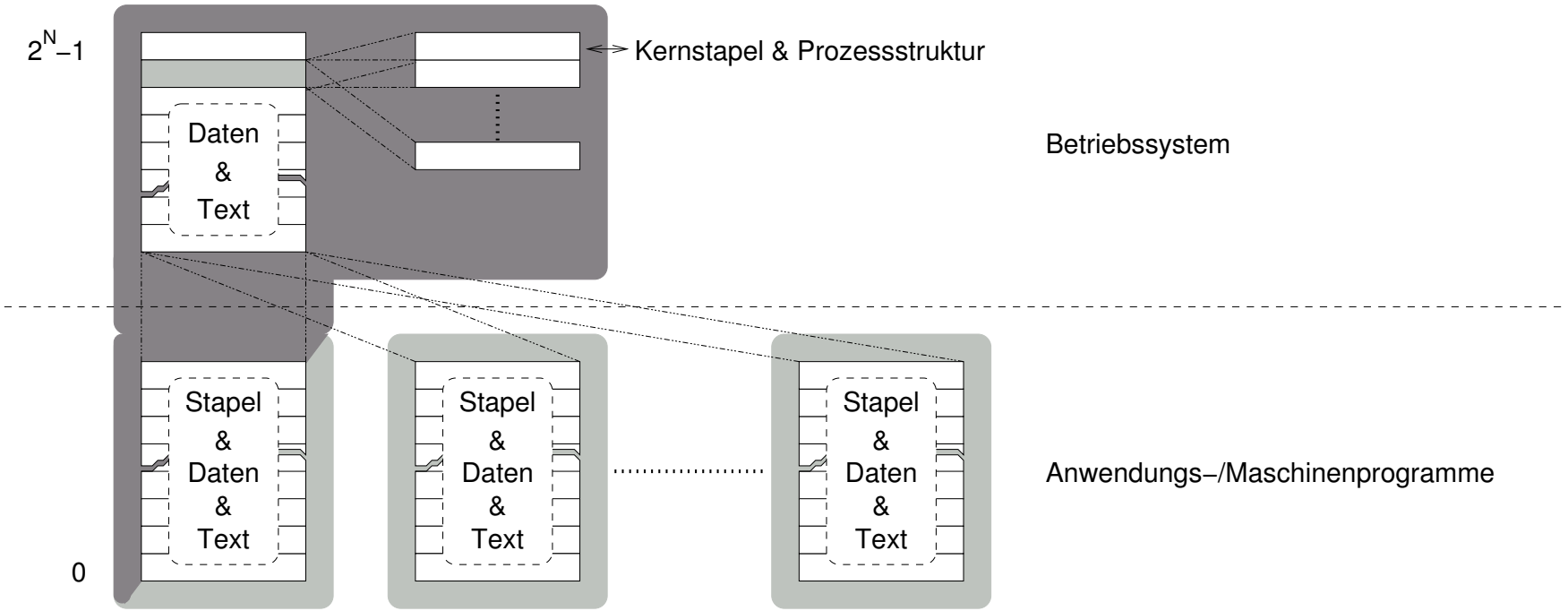
Illusion von einem eigenen physischen Adressraum bzw. -bereich für die Maschinenprogramme $\rightsquigarrow$ Inklusion des Betriebssystem(kern)s

- Vervielfachung des Adressbereichs $A_p \subset A_t$
  - A_t ist der dem Betriebssystem *total* zugeordnete Adressbereich
    - existiert einfach, aber mit A_p als integrierten variablen (mehrfachen) Anteil
  - A_p ist der einem Maschinenprogramm in A_t *partiell* zugeordnete Bereich
    - existiert mehrfach, einmal für jedes Anwendungs- bzw. Maschinenprogramm
- der Benutzeradressraum ist ein Teil des Betriebssystemadressraums
  - die MMU verhindert ein Ausbrechen von Prozessen aus A_p und A_t, nicht jedoch deren Eindringen heraus aus $A_t - A_p$ und hinein in A_p
    - bedingter Schreibschutz von A_p für A_t dämmt Betriebssystemfehler ein
  - dabei erstreckt sich A_p über den oberen oder unteren Bereich von A_t
  - ein Prozesswechsel[2] zwischen A_p bedingt das Umschalten der MMU
- im Vordergrund steht, weniger Adressraumwechsel hervorzurufen

[2] Genauer: Der Wechsel zwischen schwergewichtigen Prozessen.

Inklusion des aktiven Benutzeradressraums

gekachelt



- horizontaler und vertikaler Informationsaustausch
  - horizontal: Interprozesskommunikation, Segment-/Seitenmitbenutzung
  - vertikal: speicherabgebildeter Zugriff auf den Benutzeradressraum
- prominente Beispiele von Betriebssystemen der Art:
  - Linux: monolithisch, prozedurbasiert
  - NT: Hybridkernansatz, prozess-/prozedurbasiert

Teilung des Adressbereichs

in zweierlei Hinsicht

Inklusion des Benutzeradressraums in den Betriebssystemadressraum ist nur bei hinreichend großem N ein sinnvoller Ansatz

- das Modell wurde attraktiv mit Adressbreiten von $N \geq 30$ Bits
  - also für reale Adressbereiche ab 1 GiB Speicherumfang
- typische Aufteilung von $A = [0, 2^{32} - 1] = 4$ GiB:
  - gleich
    - 2 GiB jeweils für Benutzer- und Betriebssystemadressraum
    - NT
  - ungleich
    - 3 GiB Benutzer- und 1 GiB Betriebssystemadressraum
    - Linux, NT (Enterprise Edition)
- steht und fällt mit der Größe von Benutzerprogrammen/-prozessen

Inklusion bedeutet aber eben auch, dass die Benutzerprozesse dem Betriebssystem ein stärkeres Vertrauen schenken müssen

- Schreibschutz auf A_p legen und nur bei Bedarf zurücknehmen/lockern
- sonst sind Zeigerfehler in $A_t - A_p$ verheerend für Programme in A_p

Diskussion

- private Adressräume
  - − Systemaufrufe allein rufen bereits Adressraumwechsel hervor
  - − vertikaler Informationsaustausch ist nur indirekt möglich
  - \+ schränken den effektiv zur Verfügung stehenden Adressbereich nicht ein
  - \+ stellen geringere Anforderungen an die Korrektheit des Betriebssystems

- partiell private Adressräume
  - \+ Systemaufrufe gehen ohne Adressraumwechsel einher
  - \+ vertikaler Informationsaustausch ist direkt möglich
  - − verkleinern den effektiv zur Verfügung stehenden Adressbereich
  - − stellen höhere Anforderungen an die Korrektheit des Betriebssystems

↪ **Zielkonflikt** (*tradeoff*)

- keiner der beiden Ansätze ist *per se* ein Vorteil einzuräumen...

Gliederung

Rückblick

vgl. [2]

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

- Vergrößerung der verfügbaren Menge von Adressräumen
- Einrichtung harter Speicherschutzgrenzen
- Aufräumen, wenn Programme aussteigen, gestaltet sich einfach

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

Grundprinzip SASOS

Trennung von Belangen (*separation of concerns*)

- 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“
- 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

↪ seit Multics [3] ist beides (oft) mit dem Begriff „Prozess“ verbunden

Ewig gültige Adressen

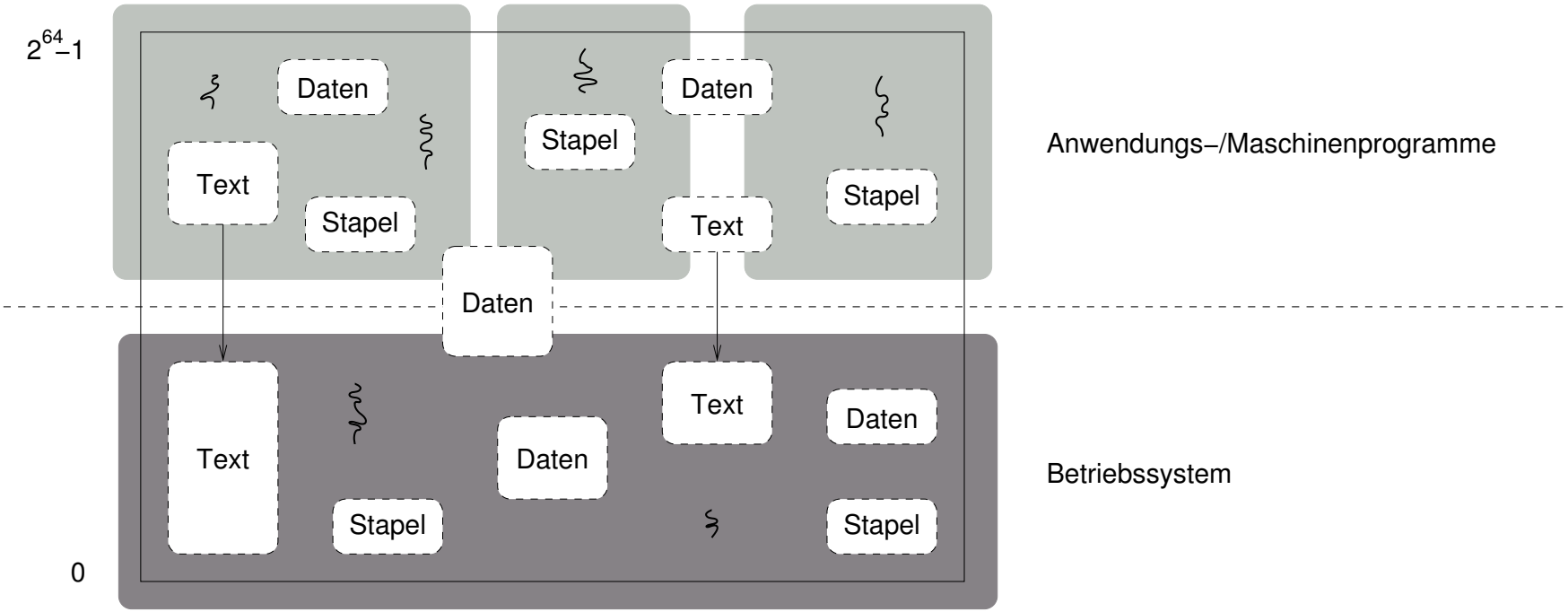
Prozessoren mit breiten Adressen fördern den Einadressraumansatz, da sie den Zwang zur Wiederverwendung von Adressen aufgeben

- 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. [2, S. 272]*

- 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
- der sämtliche im Rechensystem referenzierbare Eintitäten umfasst

„Hyperadressraum“ mit Schutzdomänen



- horizontaler und vertikaler Informationsaustausch
  - horizontal: Unterprogrammaufrufe, gemeinsame Adressbereiche
  - vertikal: *dito*
- mikrokernbasierte Beispiele von Betriebssystemen der Art:
  - Opal [2]: Mach, System- und Fernaufrufe [12] über Portale
  - Mungi [8]: L4, Interaktion nur über gemeinsame Adressbereiche

Schutz adressierbarer Entitäten

Bereiche, die 2^{64} Adressen umfassen, sind so groß, dass die einzelnen darin enthaltenen Entitäten (Objekte) nicht mehr aufzählbar sind[3]

- 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
- jede Entität ist als (seitennummeriertes) Segment ausgeprägt
  - das durch einen/mehrere Adressraumdeskriptoren repräsentiert ist
  - für alle Prozesse bspw. angelegt in einer zentralen Tabellenstruktur
- gemeinsame Nutzung setzt die Weitergabe ihrer Adressen voraus
  - dieselben Adressen selektieren denselben Deskriptor: dieselben Attribute
  - verschiedene Prozesse erhalten darüber auch nur dieselben Zugriffsrechte
- Differenzierung von Zugriffsrechten erfordert zusätzliche Maßnahmen
  - i Replikation, Anpassung und Gruppierung von Deskriptoren
  - ii Adressen mit einer **Befähigung** (*capability*, [4]) assoziieren

[3] Angenommen, dass Rechner heutiger (2015) Technologie nicht länger als etwa 584,9 Jahre im Betrieb sind – die gespeicherten Daten eingeschlossen.

Differenzierung von Zugriffsrechten

- 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

  $\hookrightarrow$ damit werden jedoch nur die typischen „Hardwarezugriffsrechte“ erfasst: zugreifen, lesen, schreiben, ausführen

- Adressen mit einer **Befähigung** (*capability*, [4]) assoziieren
  - allgemein Subjekten Rechte für Operationen auf Objekte zuschreiben [7]

    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 **!!!**

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

Fälschungssicherheit von Befähigungen

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*, [4]) 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
- 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
- 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 [16] generiert
    - z.B. thermisches Rauschen [1], durch einen A/D-Wandler dargestellt

Gliederung

Resümee

- Mehradressraumsysteme
  - bieten Virtualität durch die Vervielfachung des realen Adressbereichs
  - implementieren (total/partiell) private Adressräume
  - Informationsaustausch zwischen Betriebssystem- & Benutzeradressraum:
    - fensterbasiert, bedarfsorientierte Einblendung von Adressraumabschnitten
    - spezialbefehlbasiert, selektives Kopieren von Maschinenwörtern
    - adressraumgeteilt, direkter Zugriff von kompletten Benutzeradressraum
- 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. [8, S. 7]

Literaturverzeichnis I

[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. 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

[3] 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

[4] 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

[5] Digital Equipment Corporation (Hrsg.):
PDP-11/40 Processor Handbook.
Maynard, MA, USA: Digital Equipment Corporation, 1972.
(D-09-30)

Literaturverzeichnis II

[6] Goodenough, J. B.:
Exception Handling: Issues and a Proposed Notation.
In: *Communications of the ACM* 18 (1975), Nr. 12, S. 683–696

[7] 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

[8] 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

[9] Hildebrand, D. :
An Architectural Overview of QNX.
In: *Proceedings of the USENIX Workshop on Micro-kernels and Other Kernel Architectures (USENIX Microkernels)* USENIX Association, 1992. –
ISBN 1–880446–42–1, S. 113–126

[10] Lions, J. :
A Commentary on the Sixth Edition UNIX Operating System.
The University of New South Wales, Department of Computer Science, Australia :
`http://www.lemis.com/grog/Documentation/Lions`, 1977

Literaturverzeichnis III

[11] Lions, J. :
UNIX Operating System Source Code, Level Six.
The University of New South Wales, Department of Computer Science, Australia : http://v6.cuzuco.com, Jun. 1977

[12] Nelson, B. J.:
Remote Procedure Call.
Pittsburg, PA, USA, Department of Comupter Science, Carnegie-Mellon University, Diss., Mai 1981

[13] Parnas, D. L.:
On the Criteria to be used in Decomposing Systems into Modules.
In: *Communications of the ACM* 15 (1972), Dez., Nr. 12, S. 1053–1058

[14] Quantum Software Systems Ltd. (Hrsg.):
QNX Operating System User's Manual.
Version 2.0.
Toronto, Canada: Quantum Software Systems Ltd., 1984

[15] Schröder, W. :
Eine Familie von UNIX-ähnlichen Betriebssystemen – Anwendung von Prozessen und des Nachrichtenübermittlungskonzeptes bem strukturierten Betriebssystementwurf, Technische Universität Berlin, Diss., Dez. 1986

Literaturverzeichnis IV

[16] Wallace, C. S.:
Physically Random Generator.
In: *Computer Systems Science and Engineering* 5 (1990), Apr., Nr. 2, S. 82–88