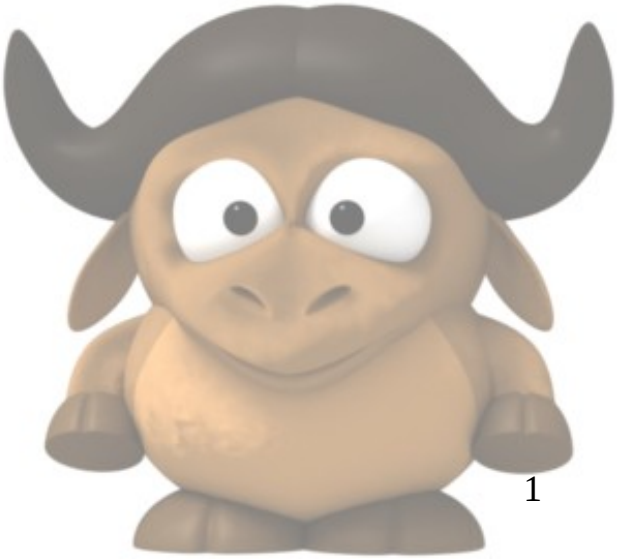
4. Der Linux-Kernel

- Geschichte des Kernels
- Die Aufgaben eines Kernels
- Kernelarten
- Implementierungsstrategien
- Bestandteile des Linux-Kernels
- Prozessverwaltung
- Prozesshierarchien
- Neuerungen im Kernel 2.6.

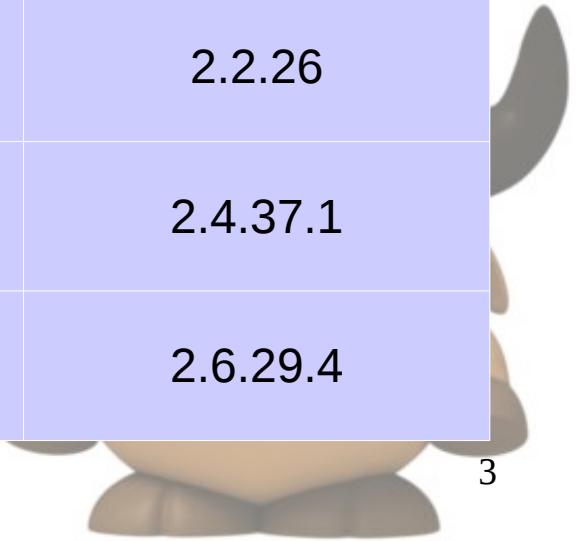


Geschichte von Linux

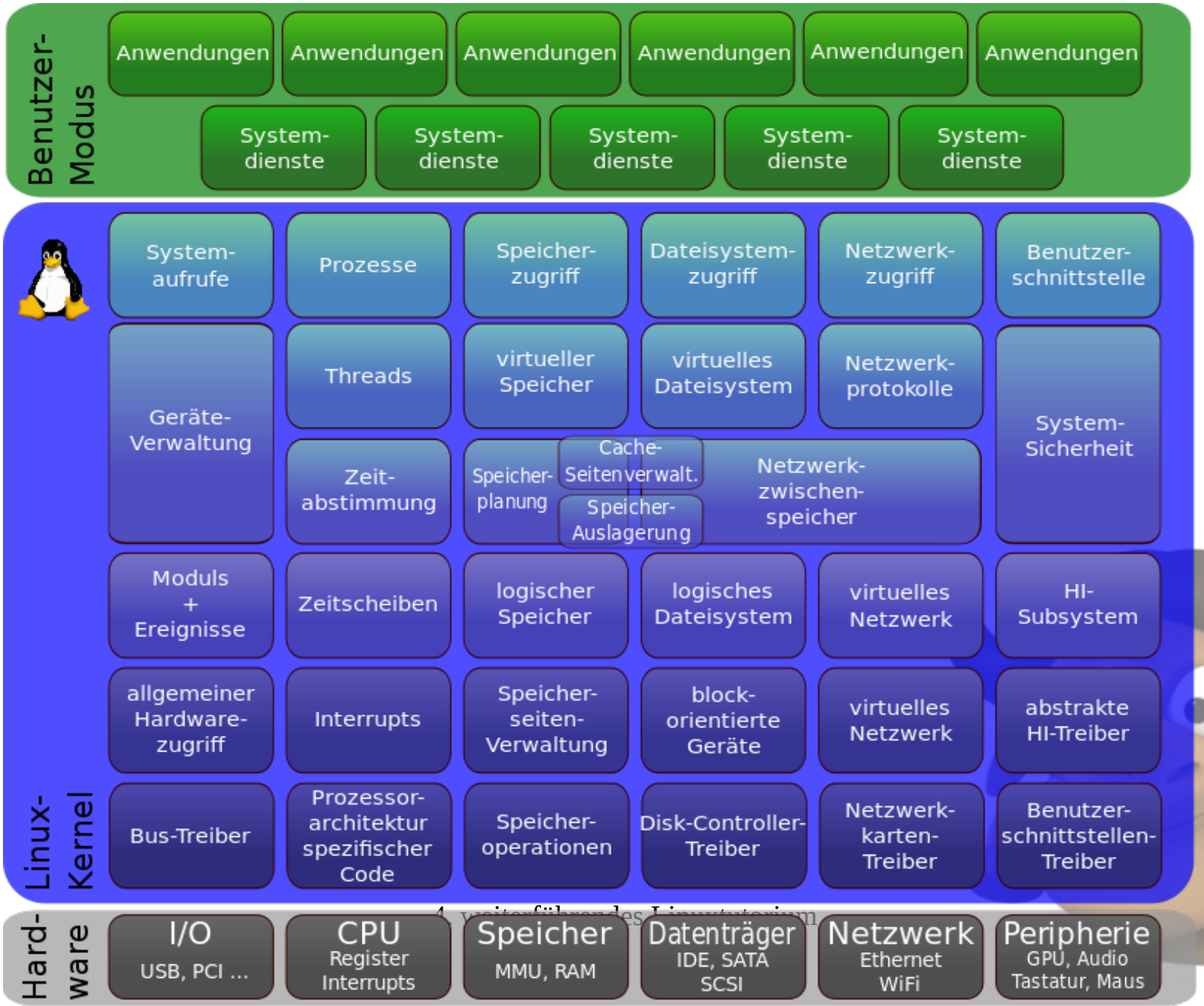
- 1991 von Linus Torvalds unter den Namen „Freaks“ freigegeben (Version 0.01)
- 1992 erfolgte die Unterstützung durch das GNU-Projekt, bei dem sich der Name GNU/Linux durchsetzte
- 1994 kam die Version 1.0 des Linux-Kernels heraus
- Der Kernel wird bis heute durch die GPL vertrieben und steht momentan unter GPL v.3
- Jede neue Stable-Version wird von Linus Torvalds freigegeben

Erscheinungstermine

Erscheinungs-datum	Version	Codezeilen	Aktuelle Version
17.09.1991	0.01	8 413	-
13.04.1994	1.0	170 581	-
09.06.1996	2.0	716 119	2.0.40
26.01.1999	2.2	1 676 182	2.2.26
04.01.2001	2.4	3.158.560	2.4.37.1
18.12.2003	2.6	5.475.685	2.6.29.4



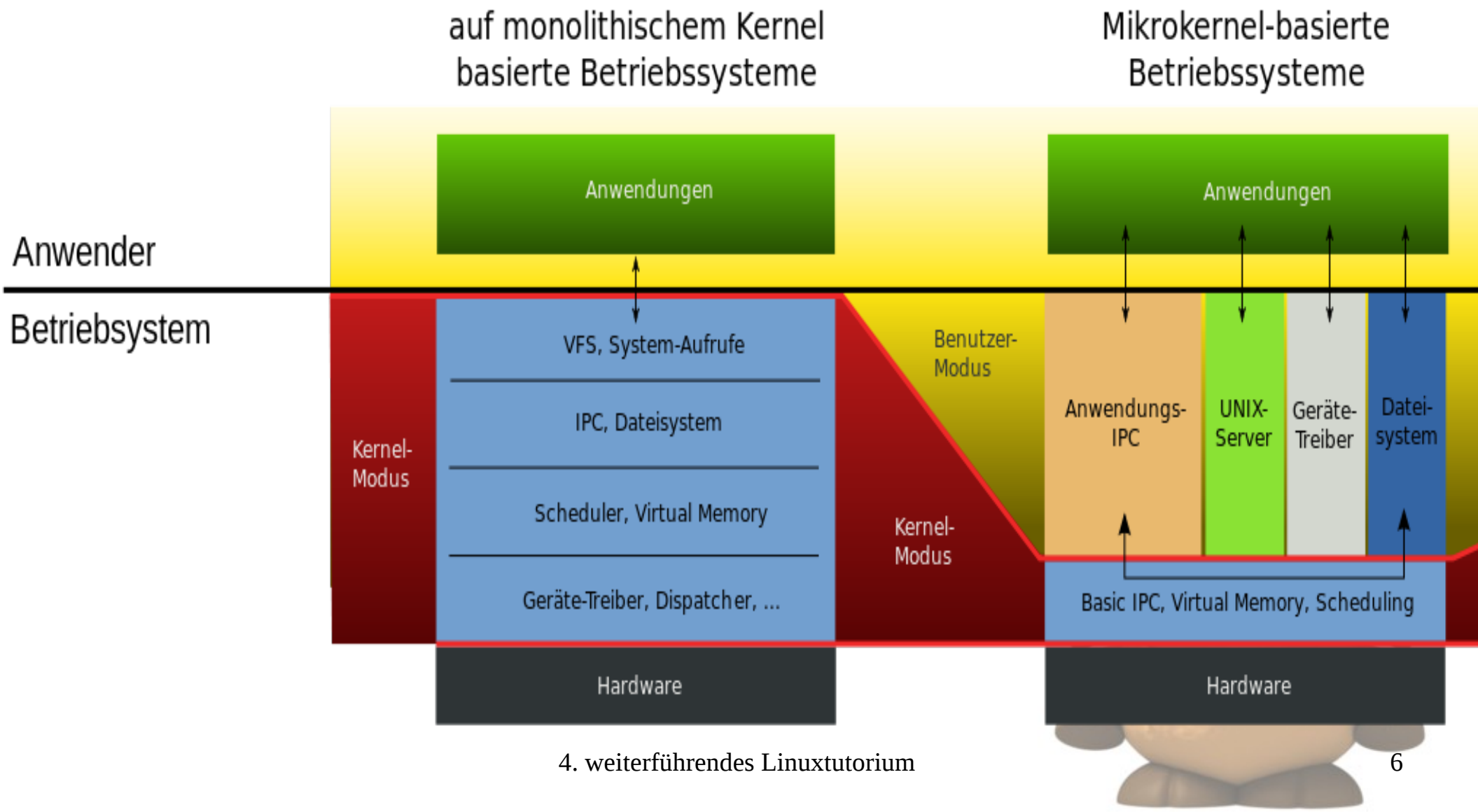
Kernelstruktur



Kernelaufgaben

- Schnittstelle zu Anwenderprogrammen
- Zugriffskontrolle auf Speicher, Peripherie, Speicher
- Verteilung der Ressourcen, z.B. Prozessorzeit auf Anwenderprogramme
- Virtualisierung der Ressourcen
- Überwachung von Zugriffsrechten auf Dateien und Geräte bei Mehrbenutzersystemen

Kernelarten



Kernelarten – monolithischer Kernel (1)

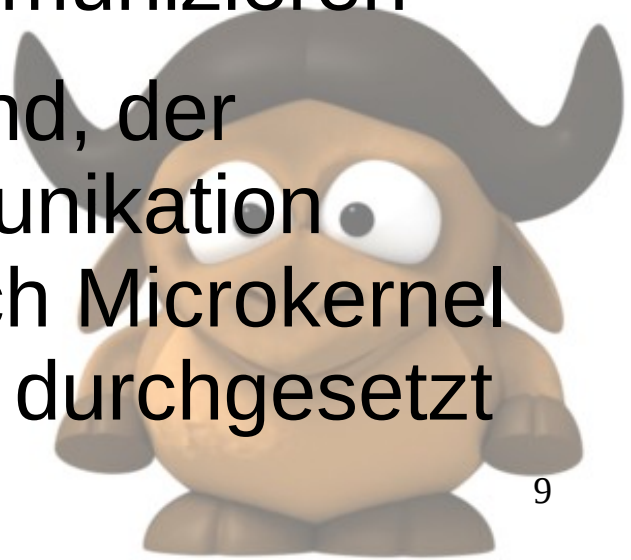
- Hier wird der gesamte Code des Kerns (Speicherverwaltung, Dateisysteme) in einer Datei verpackt
- Jede Funktion hat Zugriff auf andere Teile des Kerns
- Da die Leistung monolithischer Kernen momentan noch höher als von Microkerneln ist, wurde Linux in diesem Paradigma implementiert

Kernelarten – monolithischer Kernel (2)

- Bei Linux kommt eine wichtige Erweiterung zum monolithischen Kern zum tragen: Module
- Module enthalten ebenfalls Kernelcode
- Diese können während des laufenden Betriebs zum System hinzugefügt werden
- Durch diese Funktion gleichen monolithische Kernel Nachteile zu Microkerne aus

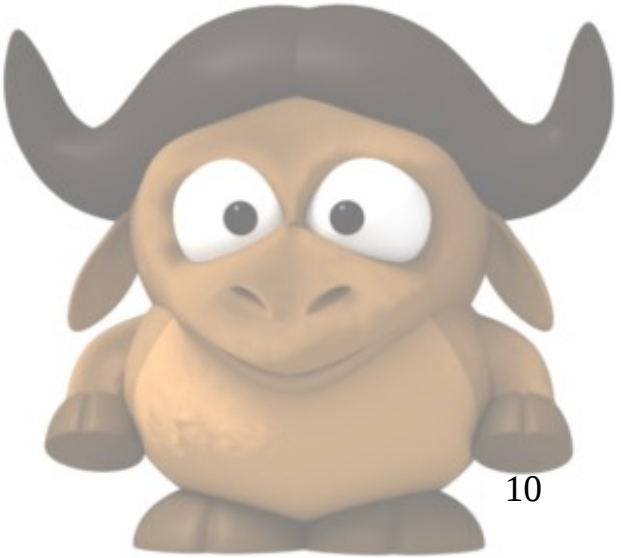
Kernelarten – Microkernel

- Nur die elementarsten Funktionen im zentralen Kernel implementiert
- Die restliche Teile werden eigenständige Prozesse ausgelagert
- Diese können nur über klar definierte Schnittstellen mit dem Kernel kommunizieren
- Durch zusätzlichen Rechenaufwand, der Aufgrund der Komponentenkommunikation verrichtet werden muss, haben sich Microkernel noch nicht allzu-sehr in der Praxis durchgesetzt



Bestandteile des Linux-Kernels

- Prozesse
- Scheduling
- Virtueller Adressraum
- Privilegstufen
- Systemcalls
- Module
- Caching
- Datentypen



Start neuer Prozesse - fork

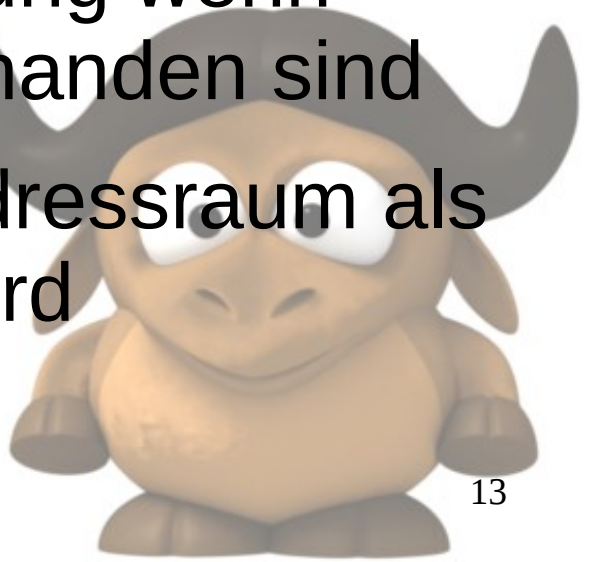
- Erzeugt eine exakte Kopie des aktuellen Prozesses, die sich lediglich in der PID unterscheidet
- Nach der Ausführung des System-Calls, existieren 2 Prozesse auf dem System
- Der Speicherinhalt des Ausgangsprozesses wird dupliziert (aus der Sicht des Programms)

Start neuer Prozesse - exec

- Dient dazu ein neues Programm in einen bereits bestehenden Content zu laden
- Die vom alten Programm belegten Speicherseiten werden gelöscht
- Danach beginnt die Ausführung des neuen Programms
- Exec erzeugt keinen eigenen Prozess, sondern kann nur einen vorher durch fork erzeugten Prozess umwandeln

Virtueller Adressraum (1)

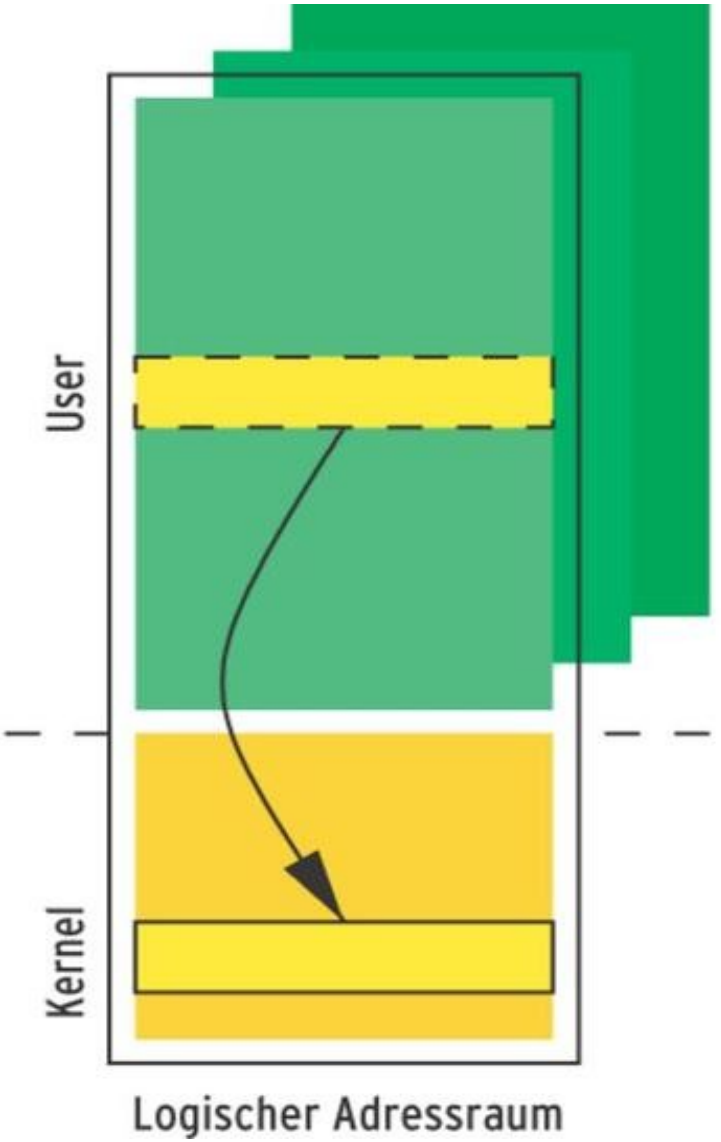
- Da Speicherbereiche über Zeiger angesprochen werden, gibt die Wortbreite dieser Zeiger, die Größe des max. verwendbaren Adressraums an
- Auf 32-Bit-Systemen Systemen sind es 2^{32}-Bit, also 4GB
- Diese 4GB stehen auch zur Verfügung wenn weniger als diese 4GB an Ram vorhanden sind
- Weshalb der max. ansprechbare Adressraum als virtueller Adressraum bezeichnet wird

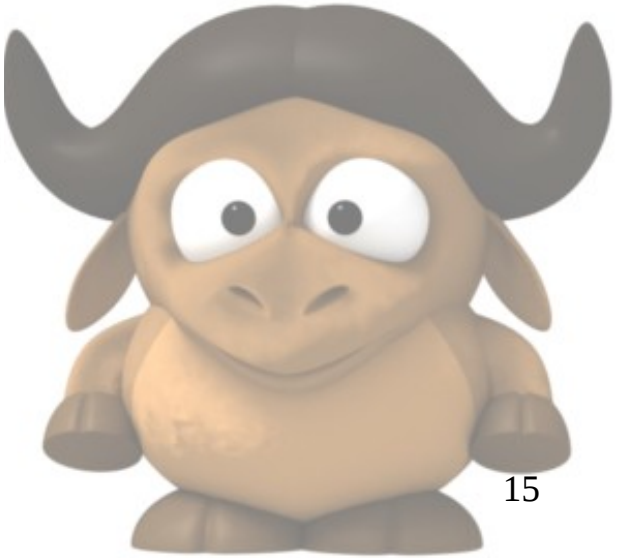


Virtueller Adressraum (2)

- Jeder Benutzerprozess eines Systems verfügt über einen virtuellen Adressbereich der von 0 bis TASK_SIZE
- TASK_SIZE ist eine Architektur-spezifische Konstante die den Adressraum in einen bestimmten Verhältnis aufteilt
- Bei 32-Bit-Systemen wir der Split bei 3GB durchgeführt

Virtueller Adressraum (3)

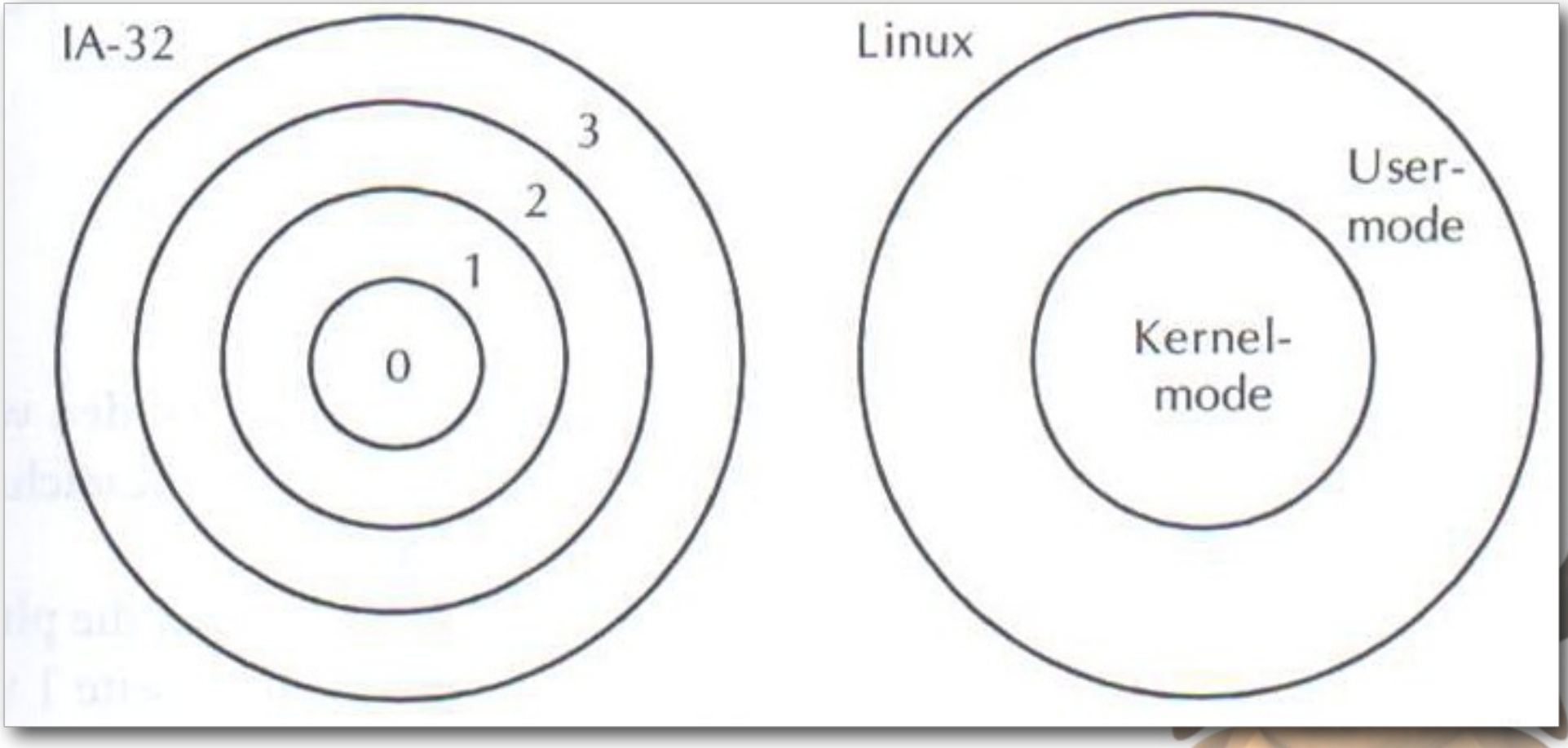




Privilegstufen (1)

- Um die einzelnen Prozesse im System vor einander schützen zu können, bietet jede moderne CPU Privilegstufen
- Jeder Prozess befindet jeweils in einer Privilegstufe
- In jeder Stufe sind verschiedene Dinge verboten
- z.B. Die Ausführung bestimmter Assembleranweisungen oder Zugriff auf bestimmte Bereiche des Virtuellen Adressraums

Privilegstufen – 32-Bit und Linux



Privilegstufen (2)

- Während moderne CPUs mehrere Privilegstufen unterscheiden können:
  - 0 - Kern
  - 1 - Systemdienste
  - 2 - Systemerweiterungen
  - 3 - Anwendungen
- Sind in Linux nur zwei Modi definiert:
  - Usermode
  - Kernelmode

Privilegstufen (3)

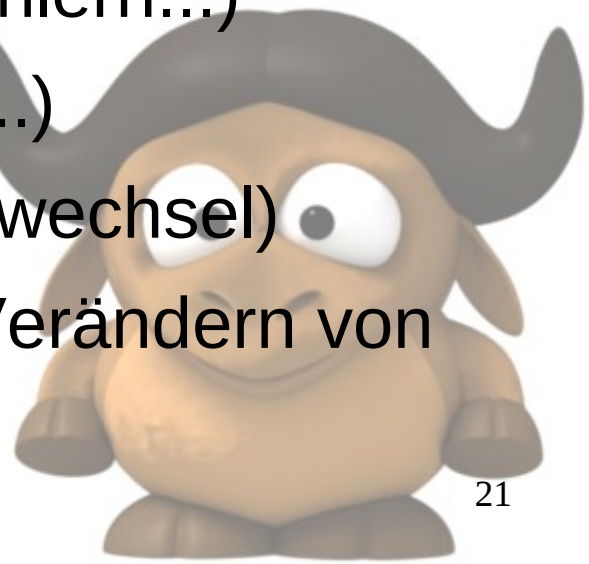
- Im Usermode ist es verboten, auf dem Speicherbereich oberhalb von TASK_SIZE, also den Kerneladressraum zuzugreifen
- Benutzerprozesse können die darin enthaltenen Daten nicht manipulieren oder auslesen
- Es ist ihnen außerdem nicht möglich, Code aus diesem Speicherbereich auszuführen
- Der Mechanismus verhindert das sich Prozesse gegenseitig stören

Privilegstufen (4)

- Der wechsel vom User- im Kernelmodus wird in einem speziellen Übergang vollzogen → Systemcall
- Wenn ein normaler Prozess-Kernelcode ausführen möchte, kann dies nur mit Hilfe eines Systemcalls geschehen
- Dieser prüft zunächst ob dem Prozess die gewünschten Aktionen gestattet sind, führt diese in seinen Namen aus und schaltet Zurück in den Usermode

Systemcalls (1)

- Sind die klassische Methode, um Userspace-Prozessen mit dem Kernel zusammenarbeiten zu lassen
- Kategorien von Systemcalls:
  - Prozessmanagement (erzeugen neuer Prozesse)
  - Signale (Senden von Signalen, Zeitzählern...)
  - Dateien (erzeugen, öffnen, schließen...)
  - Verzeichnisse (erzeugen, Verzeichniswechsel)
  - Schutzmechanismen (Auslesen und Verändern von User-ID und Group-ID)



Systemcalls (2)

- Es ist nicht möglich diese Aufgaben in einem normalen Programm unterzubringen
- Da besondere Schutzmechanismen vorhanden sind
- Außerdem müssen viele Aufrufe, auf Kernel-interne Strukturen und Vorgänge
- Nach dem Aufruf eines Systemcalls muss der Prozessor die Privilegstufe ändern

Module

- Module werden verwendet, um Funktionalitäten dynamisch zur Laufzeit des Kernels einzubinden (Gerätetreiber, Dateisysteme)
- Module sind im Prinzip nichts anderes als normale Programme, die anstatt im Usermode im Kernel-Adressraum ausgeführt werden
- Zusätzlich müssen sie bestimmte Abschnitte bereitstellen, die bei Initialisierung und Beendigung der Module ausgeführt werden

Caching

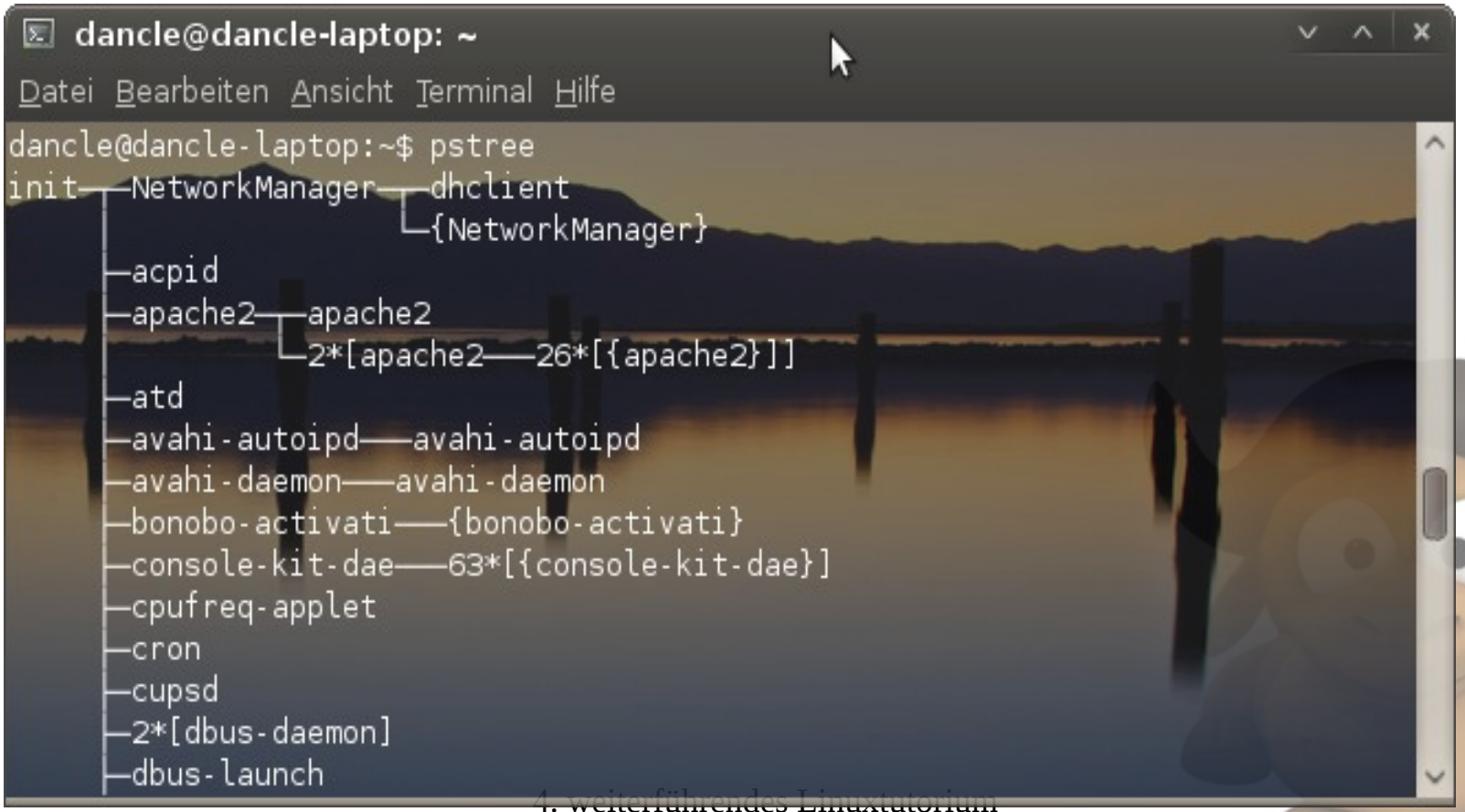
- Der Kern verwendet Caches, um die Leistung des Systems zu erhöhen
- Dies geschieht in den von langsamen Blockgeräten (Festplatten, Disketten...) gelesene Daten im Ramspeicher vorgehalten werden
- Der Cache ist Seitenweise organisiert (es werden immer komplette Speicherseiten gecacht)

Prozessverwaltung

- Prozessstruktur
- Prioritäten
- Lebenszyklus
- Repräsentation von Prozessen
- Systemaufrufe zur Prozessverwaltung
- Prozessverdopplung

Linux-Prozessübersicht (1)

- Hierarchisches Prozessschema
- Dadurch bildet sich eine Baumstruktur

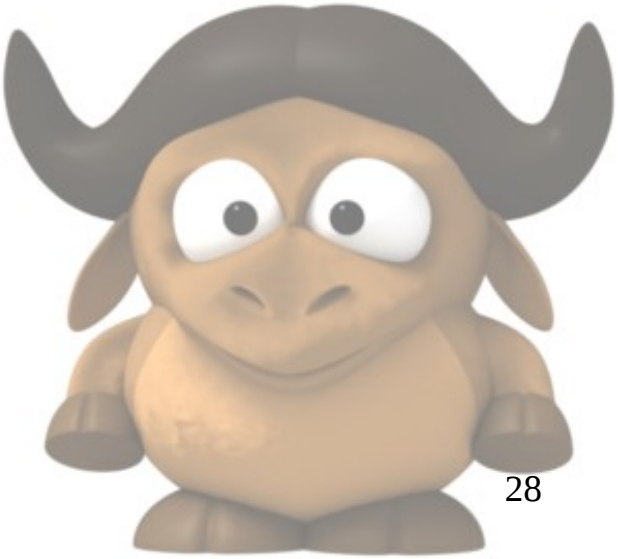


Linux-Prozessübersicht (2)

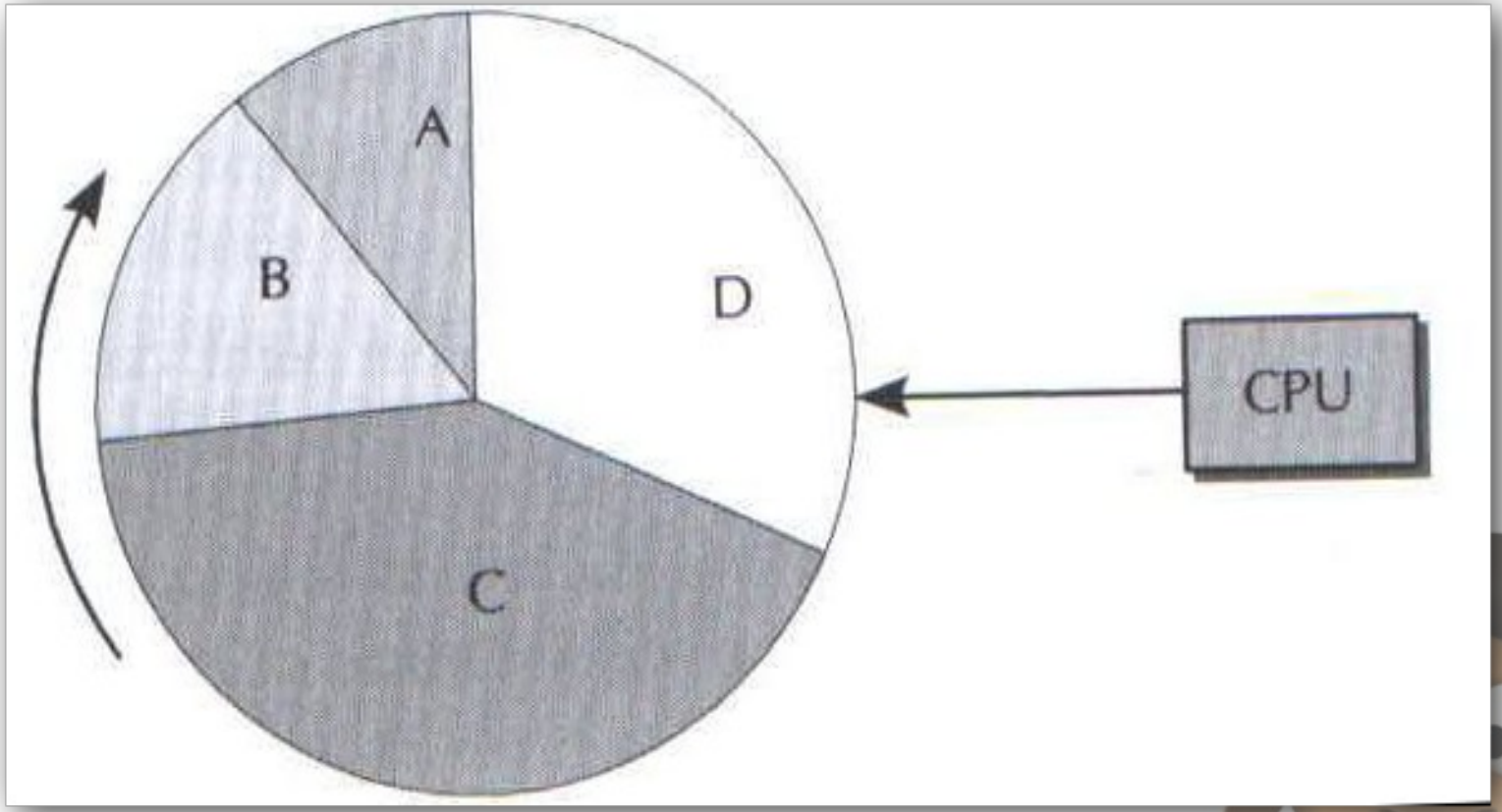
- Der Kernel startet das Programm Init als ersten Prozess
- Von dem aus die weitere Initialisierung des Systems gestartet wird
- Init ist die Wurzel aller Prozesse (des Prozessbaumes)
- Die Entstehung dieser Baumstruktur hängt eng mit der Art und Weise zusammen, wie neue Prozesse erzeugt werden

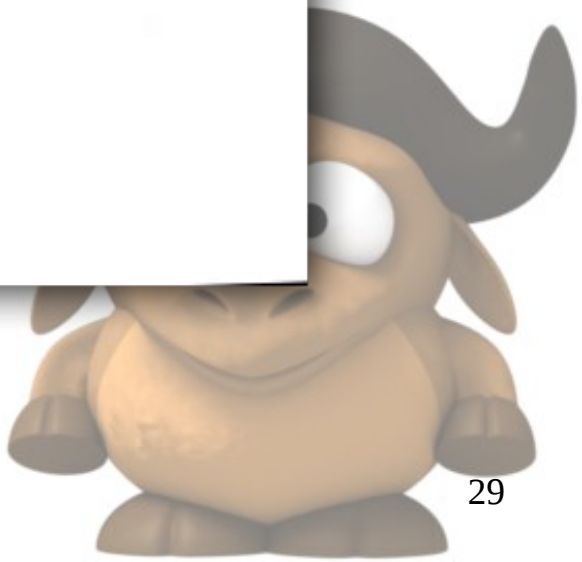
Prioritäten

- Nicht alle Prozesse auf dem System sind gleich wichtig
- Durch Prioritäten ist eine Gewichtung von Prozessen möglich
- Dadurch ist Gewährleistet das Systemprozesse bevorzugt behandelt werden



Prioritäten - Zeitscheibenverfahren

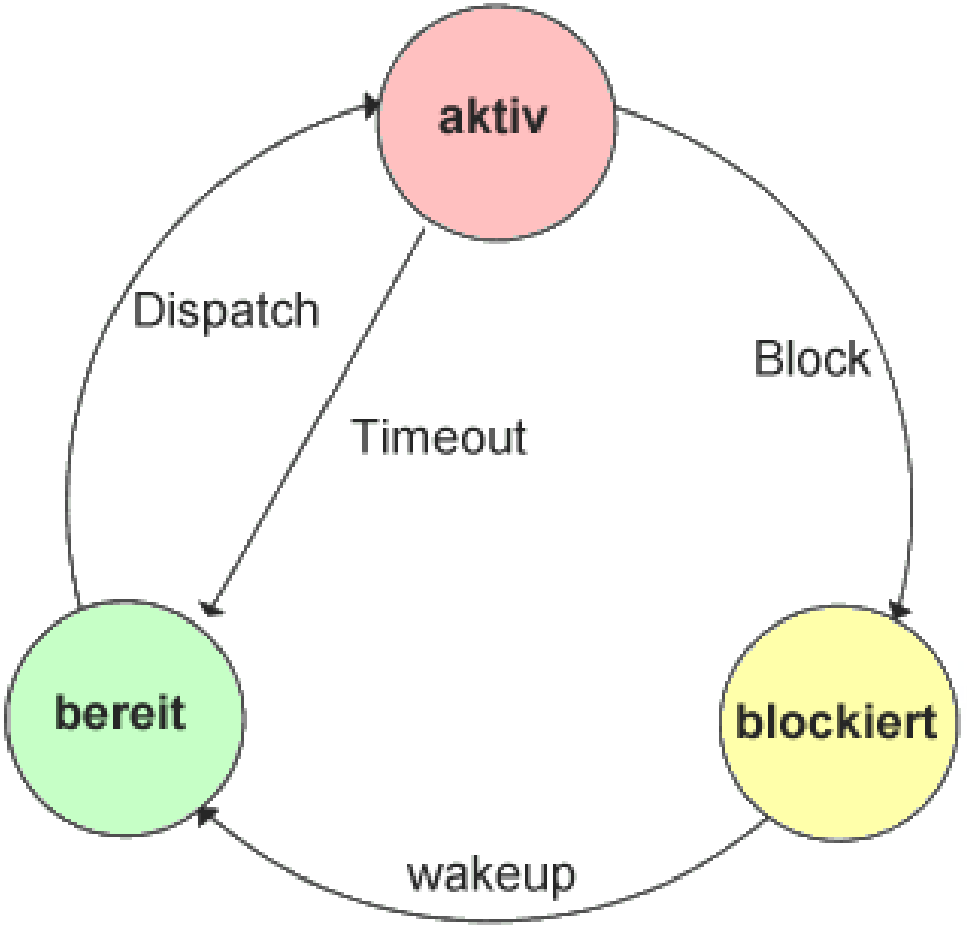


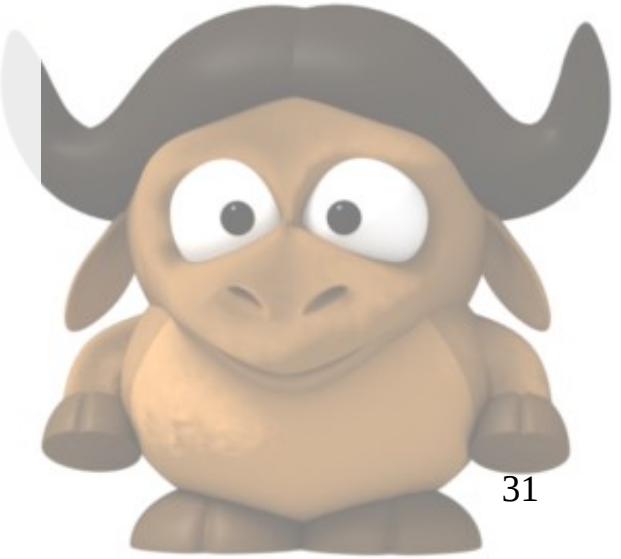


Lebenszyklus

- Zustände von Prozessen
  - Laufend – Der Prozess wird ausgeführt
  - Wartend – Der Prozess wartet auf Prozessorzeit
  - Blockiert – Der Prozess wartet auf ext. Ereignis
- Das System speichert alle Prozesse in einer Prozessliste unabhängig vom Prozesszustand
- Blockierte Prozesse sind allerdings spezielle markiert

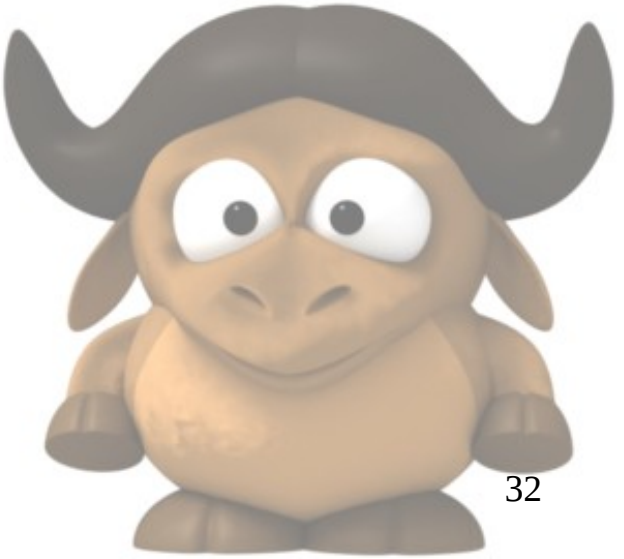
Schema des Lebenszyklus





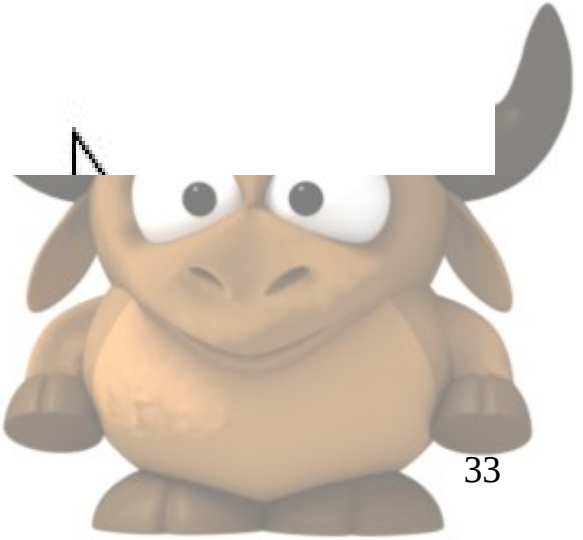
Repräsentation von Prozessen

- Alle Algorithmen im Linuxkernel die sich mit Prozessen und Programmen beschäftigen sind um eine zentrale Datenstruktur herum aufgebaut
- Es handelt sich um eine zentrale Struktur des Systems



Die Struktur task_struct

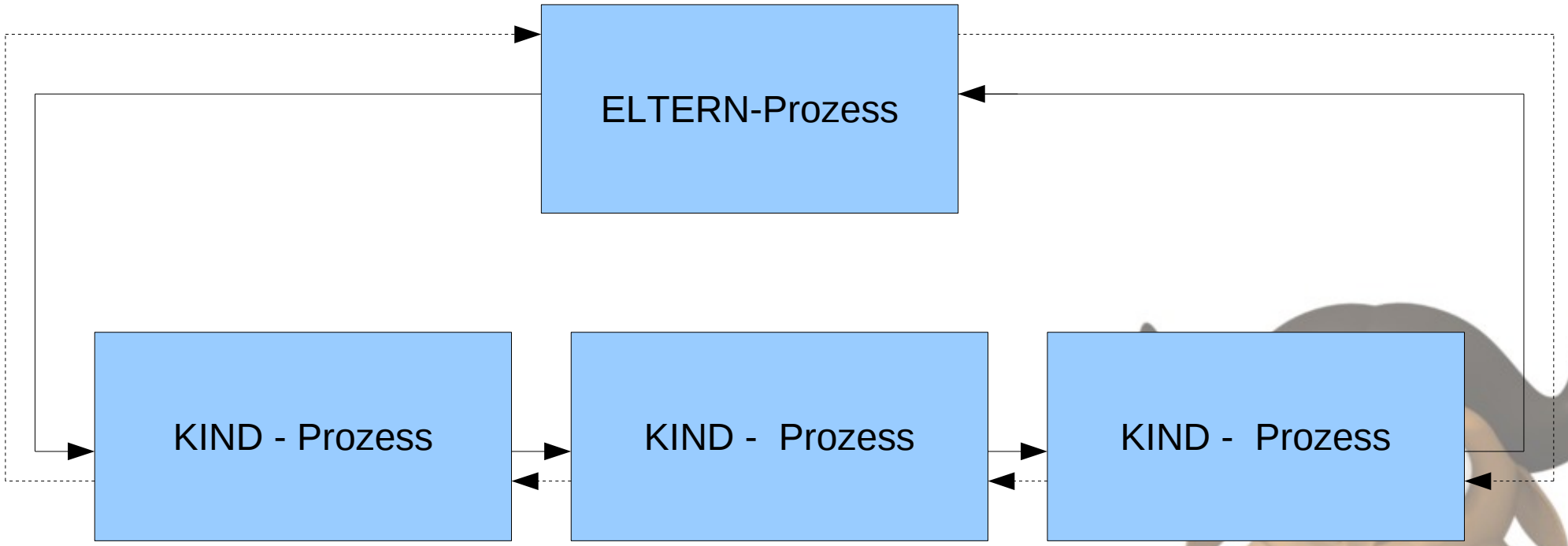
```
struct task_struct {
        volatile long state;    /* -1 unrunnable, 0 runnable, >0 stopped */
        void *stack;
        atomic_t usage;
        unsigned int flags;     /* per process flags, defined below */
        unsigned int ptrace;

        int lock_depth;         /* BKL lock depth */
```

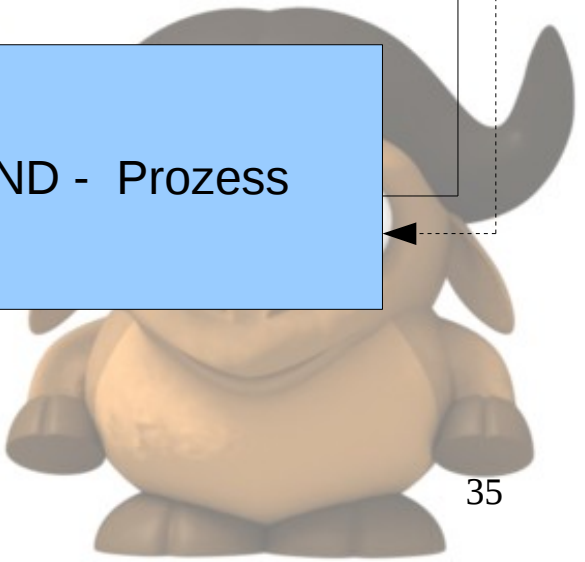


Aufbau der task_struct

- Status und Ausführungsinformationen, PID, Zeitinformationen, Pointer auf Eltern- und Geschwisterprozesse sowie Prioritätskennzahlen
- Informationen zum virtuellen belegten Speicher
- Prozesskennzahlen wie User- oder Group-ID und Sonderrechte
- Verwendete Dateien
- Threadinformationen (CPU-spezifische Laufzeitdaten)

Verwandtschaftsverhältnisse zwischen Prozessen





Prozessverdopplung (1)

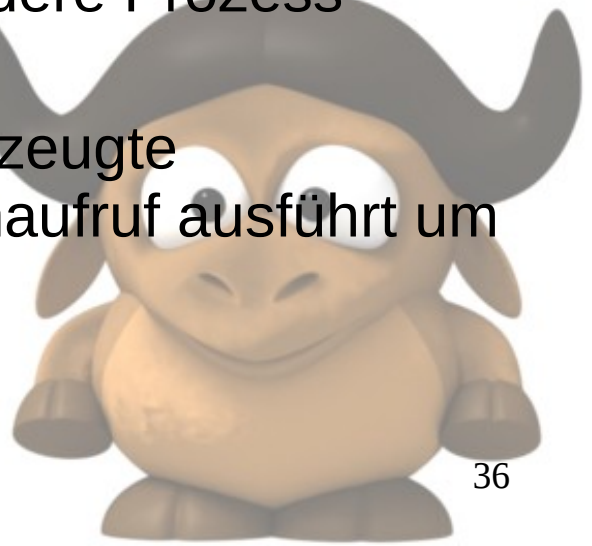
- fork

  Ist der schwergewichtigste Aufruf, da er eine komplette Kopie des Elternprozess anlegt, um dem Aufwand zu senken, verwendet Linux das COW-Verfahren.

- vfork

  arbeitet ähnlich wie fork, legt aber keine Kopie der Daten an, sondern teilt die Daten unter beiden Prozessen auf. Jeder Prozess kann Daten manipulieren, der andere Prozess merkt dies allerdings.

  Vfork ist für den Fall angelegt, dass der erzeugte Kindprozess unmittelbar den exec-Systemaufruf ausführt um ein neues Programm zu laden.



Prozessverdopplung (2)

- vfork (Fortsetzung):

  Außerdem Garantiert der Kernel, dass der Elternprozess nicht mehr ausgeführt wird, bis der Kindprozess entweder beendet wurde oder ein neues Programm gestartet hat.

- clone:

  Wird zur Erzeugung von Threads verwendet.

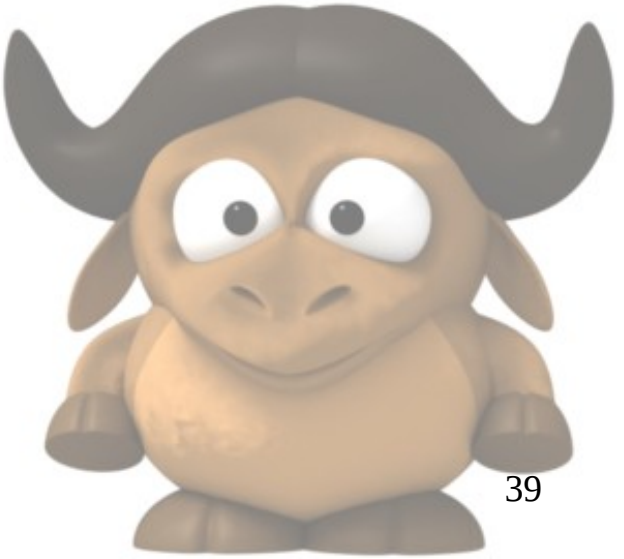
  Dabei kann genau eingestellt werden, welche Elemente zwischen Eltern- und Kindprozess geteilt werden und welche Kopiert werden.

Copy-On-Write

- Wird benutzt um das Kopieren aller Daten des Elternprozesses bei der Ausführung von fork zu vermeiden
- Sie beruht auf der Erkenntnis, das Prozesse nur einen geringen Teil der Speicherseiten ausnutzt
- Solange beide Prozesse nur lesend auf Ihre Speicherseiten zugreifen, können die Daten problemlos geteilt werden
- Sobald ein Prozess versucht die geteilte Seite zu schreiben, wird Sie exklusiv für ihn kopiert

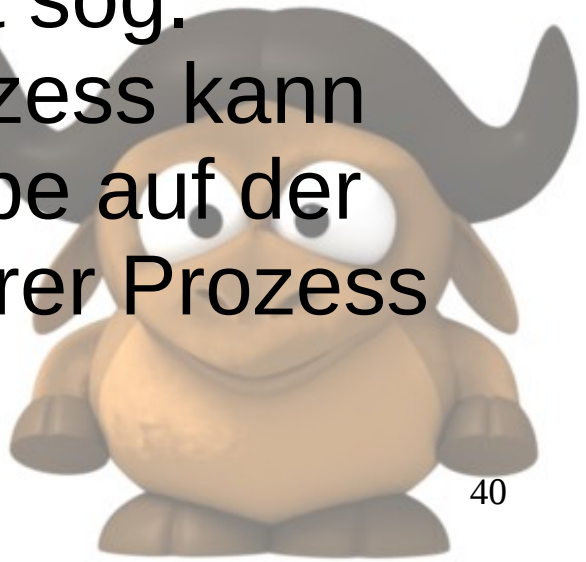
Neuerungen im Kernel 2.6

- Der O(1)-Scheduler
- Der CFS-Scheduler
- Preemptiver Kernel
- Zugriffskontrolllisten
- Inotify
- Weitere wichtige Änderungen



Der Scheduler

- Scheduler = Steuerprogramm, der die zeitliche Ausführung mehrere Prozesse regelt
- Non-preemptive lassen ein Prozess solange laufen, bis dieser von sich aus die CPU wieder freigibt
- Preemptive Scheduler arbeiten mit sog. Timeslices (Zeitscheiben), ein Prozess kann max. für die Dauer einer Zeitscheibe auf der CPU laufen, danach wird ein anderer Prozess ausgewählt



Anforderungen an den Scheduler durch verschiedene Systemarten

- Beispiele für Systemarten:
  - Batch Systeme:

    Keine hohe Anforderung an die Reaktionszeit
  - Interaktive Systeme:

    möglichst kurze Reaktionszeit gewünscht
  - Echtzeitsysteme (Realtimescheduling):

    hartes Zeitlimit für die max. Bearbeitungsdauer eines Prozesses

Ziele des Scheduler

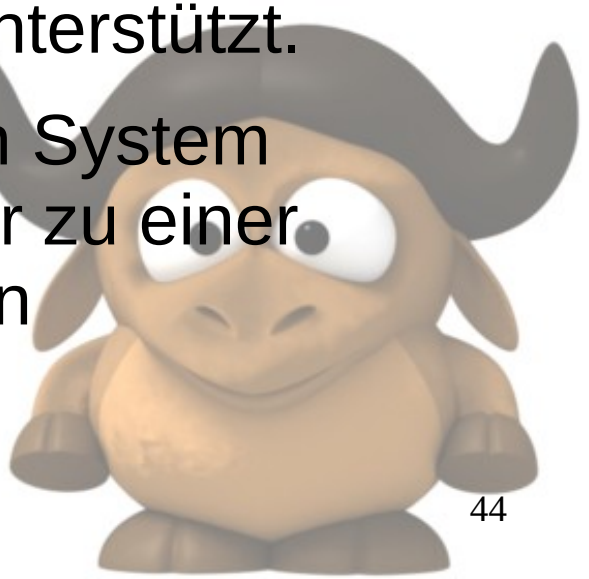
- Der Scheduler ist bestrebt, möglicht allen Anforderungen der Systemarten gerecht zu werden
- Allgemeine Ziele: Fairness und Balance
- Batch Systeme: hohe Ø-CPU-Last, hoher Durchsatz
- Interaktive Systeme: schnelle Antwortzeiten
- Echtzeitsysteme: Deadlines einhalten

Der O(1) - Scheduler

- Wurde mit Kernel 2.6. eingeführt, aber ab Version 2.6.23 von Completely Fair Scheduler abgelöst
- Er erhielt seinen Namen durch die Komplexität 01, das bedeutet, das die vom Scheduler benötigte Zeit um einen nächsten Prozess auszuwählen, unabhängig von der Anzahl der verwalteten Prozessen war

Completely Fair Scheduler

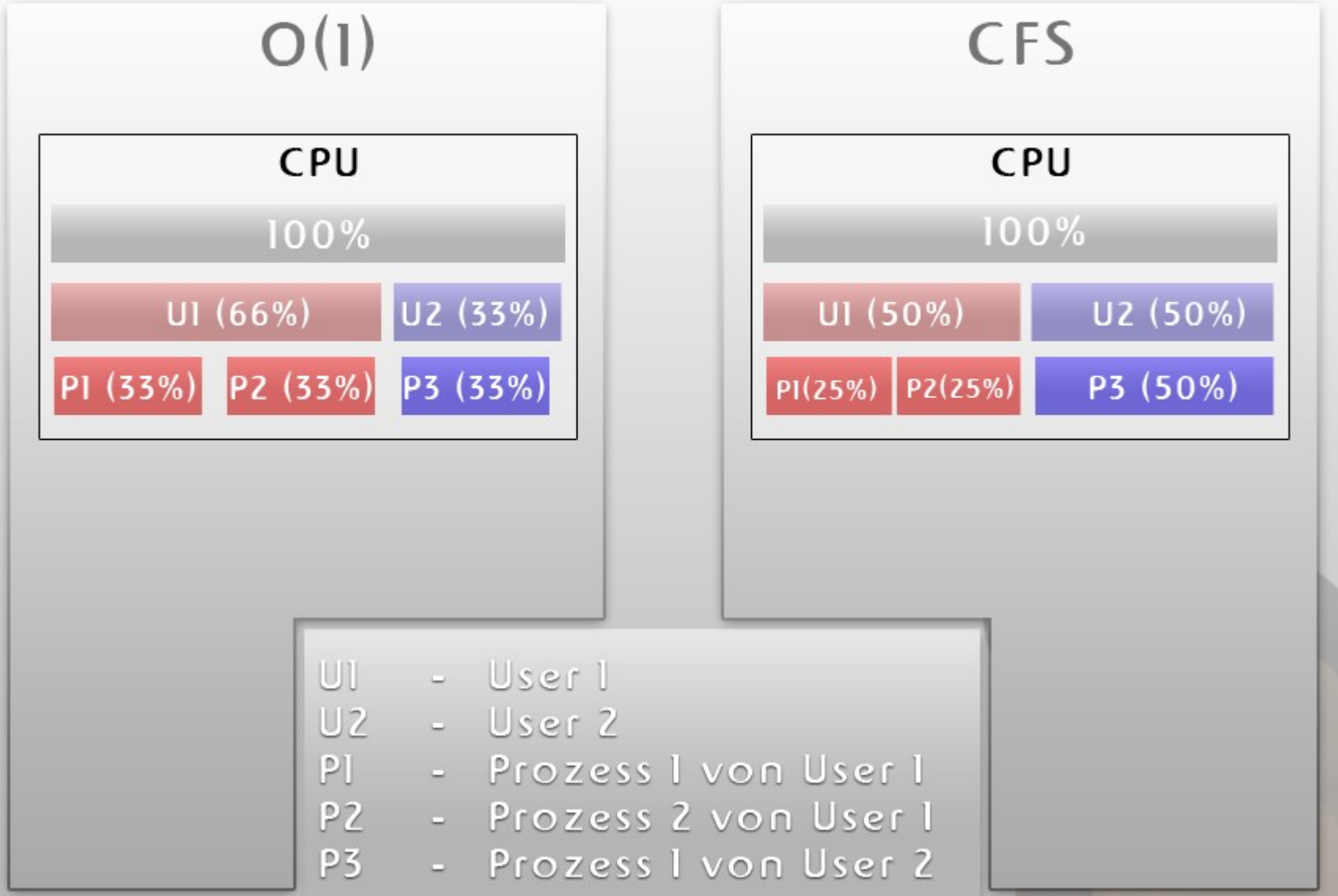
- Der Completely Fair Scheduler garantiert eine faire Aufteilung der Prozessorzeit
- Er verzichtet im Gegensatz zu O(1)-Scheduler auf Statistiken und Heuritiken
- Im Idealfall läuft beim CFS jeder Task quasiparallel in gleicher Geschwindigkeit
- Ab Kernel 2.6.24 wird CFS-Taskgroups unterstützt.
- Sinnvoll wenn mehrere Benutzer an einen System arbeiten, da die Tasks von einen Benutzer zu einer Gruppe zusammengefasst werden können



CFS-Funktionsweise

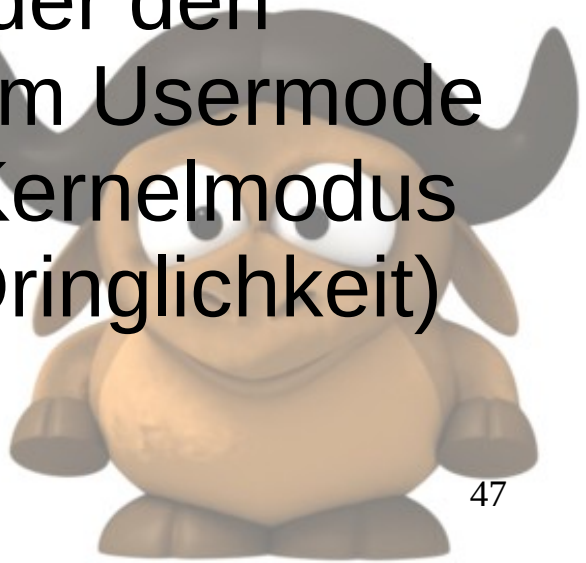
- Der CFS kennt keine Timeslices und keine Runqueue
- Stattdessen ist jedem Prozess ein wait_runtime-Wert zugeordnet
- Dieser bestimmt auf Nanosekundengenau wielange der Prozess auf seine Ausführung wartet
- Der Prozess mit höchster wait_runtime wird gewählt

Die Scheduler ab Kernel 2.6



Preemptiver Kernel

- Preemptives Scheduling ist jetzt auch im Kernelmodus möglich
- Somit kann ein Programm aus dem Usermode, wenn es eine dringende Anfrage hat, ein Programm aus dem Kernelmode unterbrechen
- Der Kernelmodus nimmt dann wieder den Betrieb auf, wenn die Timeslice vom Usermode -Prozess abgelaufen ist oder der Kernelmodus ein Re-Scheduling anfordert (mit Dringlichkeit)



Zugriffskontrolllisten

- Mit dem Kernel 2.6 werden für Linux erstmals Zugriffskontrolllisten (access control lists) nativ eingeführt
- Eine ACL legt fest, welcher Benutzer welche Dienste und Dateien nutzen darf
- Im Unterschied zu einfachen Zugriffsrechten sind ACLs feiner einstellbar
- Unter Linux unterstützen dabei die Dateisysteme ext2, ext3, JFS, XFS und ReiserFS ACLs vollständig

Inotify

- Mit dem Kernel 2.6.13 hält erstmals Inotify Einzug in den Kernel
- Dies ermöglicht das permanente Überwachen von Dateien und Ordnern
- Wird eines der überwachten Objekte geändert oder ein neues Objekt im Überwachungsraum erschaffen, gibt Inotify eine Meldung aus
- Dies wiederum kann andere Programme zu definierten Tätigkeiten veranlassen kann

Weitere wichtige Änderungen

- In Linux 2.6 wurde die Maximalzahl für bestimmte Ressourcen angehoben
- Die Anzahl von möglichen Benutzern und Gruppen erhöhte sich von 65.000 auf über 4 Milliarden
- Ebenso wie die Anzahl der Prozess-IDs (von 32.000 auf 1 Milliarde)
- Und die Anzahl der Geräte (Major/Minor-Nummern)
- Außerdem wurde für die Verwaltung der I/O-Gerätedateien devfs durch udev ersetzt, was Unzulänglichkeiten, wie z.B. ein zu großes /dev/-Verzeichnis, beseitigt