



Enterprise Computing

Unix und Linux auf System z

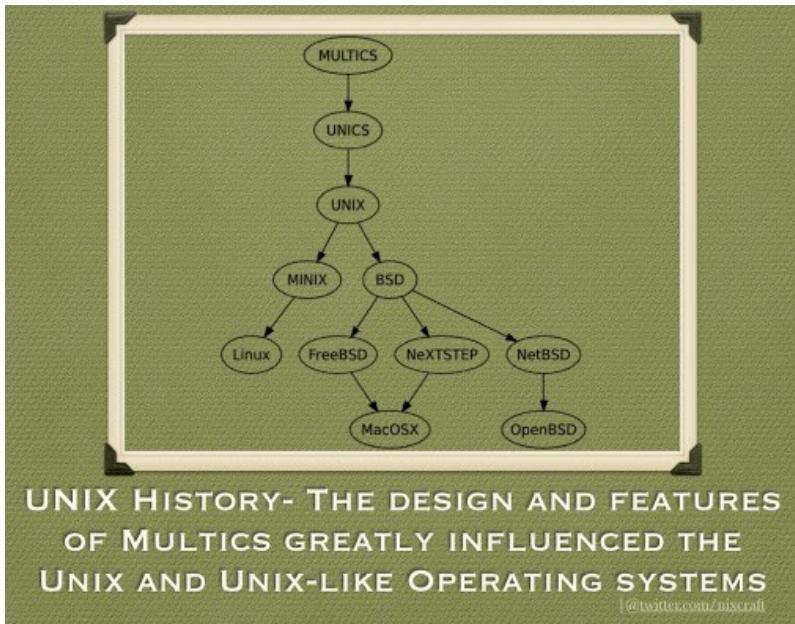
Prof. Dr.-Ing. Wilhelm G. Spruth
Dipl. Inf. Gerald Kreißig

WS2016/17

Unix und Linux auf System z Teil 1

UNIX

Entstehung der Unix basierten Betriebssysteme



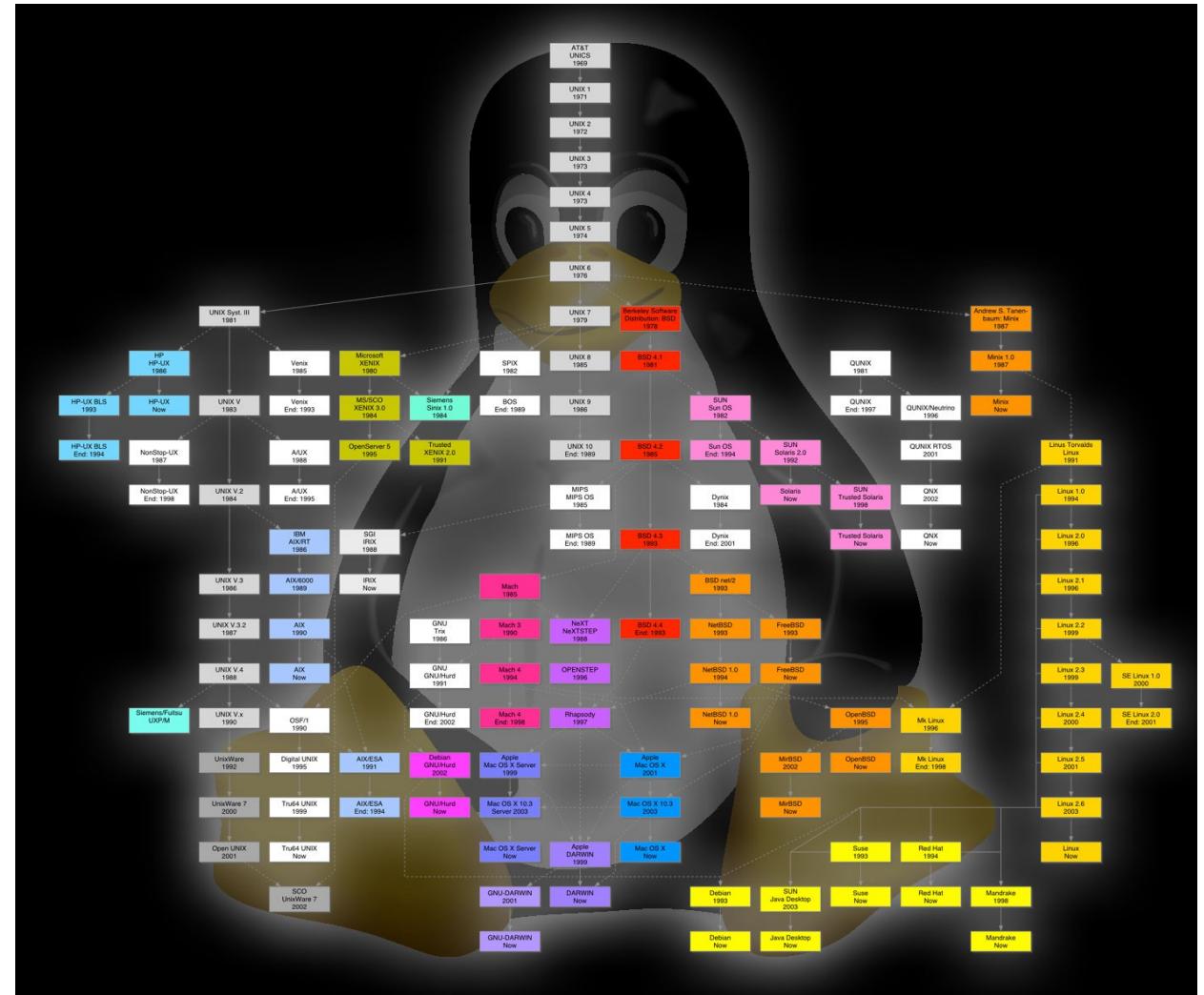
Dazu gehören die Serverprodukte

- UTS von Amdahl,
- HP-UX von HP,
- Solaris von Sun/Oracle,
- AIX von IBM,

PC Produkte wie Mac OS und die verschiedenen OpenSource Systeme von Linux

Unix selbst basiert auf dem
Multics Betriebssystem!

Unter Unix basierten Betriebssystemen versteht man die Betriebssysteme, die von der ursprünglichen Entwicklung aus den Bell Labs ableitbar sind.



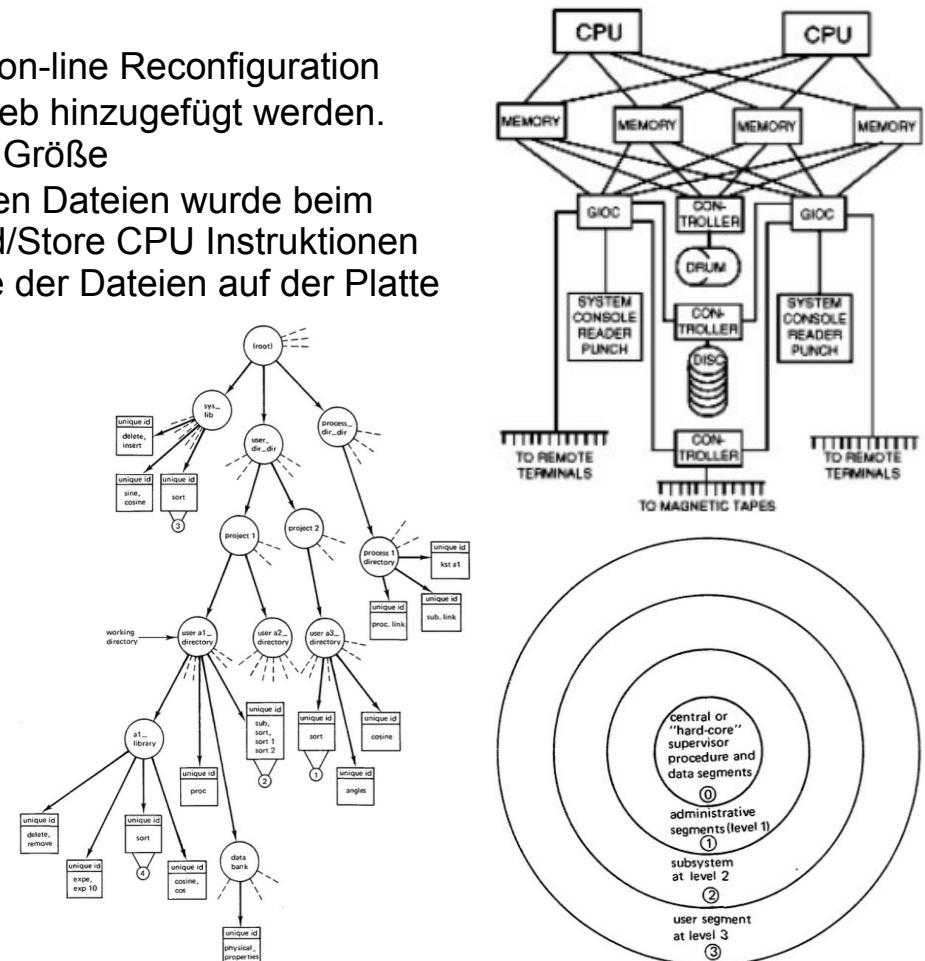
<http://www.techieapps.com/the-history-of-linux-shaping-of-the-penguin-to-its-despondency>

Multics - Multiplexed Information and Computing Service

Multics wurde am **Massachusetts Institute of Technology (MIT)** von 1964 bis 1969 (1. Release) entwickelt als gemeinsames Projekt mit General Electric und Bell Labs (Forschungseinrichtung von AT&T). Das Ziel war, alle bis dahin bekannten Betriebssystemkonzepte aus der Forschung in ein reales System umzusetzen. Heraus kam ein revolutionäres System mit vielen „state-of-the-art“ Implementierungen:

- Unterstützung von **Multiprocessor Systemen** mit sehr flexibler on-line Reconfiguration CPUs, Arbeitsspeicher, I/O einheiten konnten im laufenden Betrieb hinzugefügt werden.
- **Virtual Single-Level Memory** mit Speichersegmenten variabler Größe
Der gesamte Speicher eines Prozesses inklusive den persistenten Dateien wurde beim Zugriff in den Adressraum aufgenommen und mit normalen Load/Store CPU Instruktionen bearbeitet. Das Betriebssystem sorgte dafür, dass die Segmente der Dateien auf der Platte gesichert wurden.
- **Hierarchisches Filesystem** mit Dateinamen beliebiger Länge und Syntax. Directories konnten auf Dateien verweisen oder auf andere Directories oder konnten symbolische Links enthalten. Damit konnte jede Directory / Datei mehrere Namen besitzen.
- **Capability based Architecture** mit Zugriffslisten (access lists) zu jeder Datei erlaubte flexibles File-Sharing, aber auch vollständige Exklusivität des Dateizugriffes.
- **Zugriffsringe (Security Rings)** kontrollierten die Datensicherheit während der Ausführung eines Prozesses.

Multics wurde ursprünglich für das **GE-645 Mainframe System** entwickelt und später auf die **Honeywell 6180 Serie** portiert. Es war bis zum Jahr 2000 im Einsatz.



UNIX Time-Sharing System

Das Design und die Eigenschaften von Multics hatten großen Einfluß auf die Entwicklung von Unix, das von 2 Programmierer **Ken Thompson** und **Dennis Ritchie** geschrieben wurde, die auch schon an dem ursprünglichen Multics Project mitgearbeitet hatten. Beide waren Mitarbeiter von **Bell Labs**, dem Forschungslabor der amerikanischen Telefongesellschaft AT&T.

Auf den ersten Blick sehen viele Eigenschaften von Unix wie bei Multics aus (z.B. das hierarchische Filesystem, die Commandnamen, etc.) aber die interne Designphilosophie war ganz anders und darauf fokussiert, das Betriebssystem klein und einfach zu gestalten. Damit wollte die beiden Entwickler einige der Nachteile von Multics umgehen, die sehr hohen Resourcebedarf hatten und nur von einem teuren Mainframe erfüllt werden konnte.

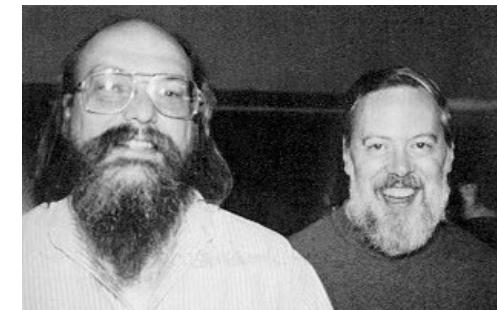
Der Name *Unix* (ursprünglich Unics) ist ein Wortspiel bzgl. *Multics*. Es heißt, dass das *U* in Unix für *uniplexed* steht im Gegensatz zu *multiplexed* von Multics, und damit die Intention der Entwickler unterstreicht, die Komplexität von Multics aufzugeben zugunsten eines einfacheren funktionierenden Betriebssystem für kleine Computer.

Die Entwickler heben folgende Eigenschaften von Unix besonders hervor:

[Communications of the ACM July 1974 Volume 17 Number 7](#)

It offers a number of features seldom found even in larger operating systems, including:

- (1) *a hierarchical file system incorporating demountable volumes;*
- (2) *compatible file, device, and inter-process I/O;*
- (3) *the ability to initiate asynchronous processes;*
- (4) *system command language selectable on a per-user basis;*
- (5) *over 100 subsystems including a dozen languages.*



Ziel war ein Time-Sharing Betriebssystem für die PDP Mini-Computer von Digital Equipment DEC, zuerst auf einer PDP-8 und dann vor allem auf der PDP-11, einem sehr erfolgreichen System zur Prozessautomatisierung.

Später wurde das System innerhalb von AT&T auf weitere Hardware portiert, unter anderem auf die Interdata 8/32, mehreren Realtime Systemen und S/370 Mainframes. Nach und nach wurde es für alle AT&T Computer Projekte eingesetzt.

Universitäten und andere Forschungseinrichtungen konnten relativ günstig eine Source-Code Lizenz erwerben, was dazu führte, das Unix an vielen Universitäten eingesetzt wurde und zur Basis von Betriebssystem-Forschungsprojekte wurde.

Unix System Konzepte

Unix hat auf Grund seiner Multics Abstammung viele Konzepte realisiert, die dem aktuellen Stand der Forschung entsprachen.

Prozessverwaltung:

Unix unterscheidet zwischen Image und Process – ein **Image** ist der aktuelle Zustand eines ausführbaren Programmes mit allen Speicher- und Registerinhalten sowie den geöffneten Dateien. Ein **Process** ist ein Image in Ausführung. Mittels dem `fork()` Systemcall wird das Image des aktuellen Process als Child-Process gecloned und es erbt alle Zustände des Parent-Process. Mit dem `exec()` Systemcall wird dem Process ein neues Programm zugewiesen. Process-to-Process Kommunikation wird mit Hilfe von Pipes im Shared memory durchgeführt.

Hierarchisches Filesystem:

Es wird zwischen **ordinary files**, **directories** und **special files** unterschieden.

- **Ordinary Files** enthalten die Benutzerdaten als reinen Bytestream. Es wird keine interne Struktur vom Betriebssystem vorgegeben, sondern sie wird von der Anwendung kontrolliert.
- **Directories** enthalten die Verweise auf die anderen Files. Jedes Directory verhält sich wie ein ordinary file, aber es kann nur von speziellen Programme geschrieben werden. Ein spezielles Directory ist **root /** als oberstes Element.
- Alle unterstützten I/O Geräte (Terminal, Drucker, Plattenspeicher, etc.) werden durch mindestens einen **Special File** repräsentiert. Alle Special Files befinden sich im `/dev` Directory.
Der Vorteil dieses Konzeptes ist, dass Datei- und I/O Operationen die gleichen sind, die Namensgebung ist gleich und es gelten dieselben Schutzmechanismen.

Für alle Elemente des Filesystems gelten die gleichen **Zugriffsrechte** rwx für die Benutzergruppen user/group/world sowie als 10-tes Protection Bits das Recht set-user-id

Command Line Interface

Jeder Benutzer hat die Möglichkeit, sein eigenes Terminal Interface mit Hilfe einer Command Line Shell auszuwählen und zu personalisieren. Die standard **Shell bash** erlaubt es, Programme im Vorder- oder Hintergrund auszuführen, Eingabe und Ausgabe zu re-directen und programme mittels Pipes miteinander kommunizieren zu lassen.

Unix Komponenten

Ritchie / Thompson sagen:

„*we are programmers, we naturally designed the system to make it easy to write, test, and run programs.*“

Dazu gibt es eine Vielzahl von Funktionen und Programmen, die dieses Ziel unterstützen.

Als Grundregel für Kommandos und Applikationen gilt, dass sie sich auf eine Funktion konzentrieren und diese richtig machen.
(Beispiel: die Compiler übersetzen die Programme, aber formattieren die Listings nichts. Dafür gibt es eigene Programme)

Kommandos:

- find – finde eine Datei innerhalb des Directorybaumes
- grep – durchsuche eine Datei nach einer spezifizierten Zeichenkette
- ls – zeige den Inhalt von Directories
- usw.

Text Editor – ursprünglich nur Zeilen-orientierte Editoren wie ed oder Streaming Editor sed.
Full-screen Editoren wie vi kamen erst später dazu

C Programming Language

Dutzende von Programmiersprechen
Fortran 77 / Basic / Snobol / APL / ALGOL 68 / Pascal / ...

Programmentwicklungsumgebung

- sccs – Source Code Control System zur Verwalten von Programmversionen
- yacc – Yet Another Compiler Compiler zur Erstellung von Compiler, insbesondere der syntaktischen Analyse
- lex – erzeugt Scanner für die lexikalische Analyse von Quelltexten

Dokumentenerstellung

- nroff / troff – programmierbare Textformattierer mit Makros zur Steuerung der Textdarstellung
(Vorgänger zu Textverarbeitungsprogrammen wie LaTex oder HTML/CSS)

UNIX Weiterentwicklungen

Auf Grund der relativ großzügigen Lizenzkonditionen von AT&T für Universitäten entstanden eine Reihe von interessanten Unix Versionen.

Berkeley Unix wurde an der UC Berkeley entwickelt und führte wesentliche Erweiterung ein:

- Das **Fast File System**: ein schnelleres Dateisystem mit langen Dateinamen (die Edition 7 erlaubte nur 14 Zeichen).
- Die **Internetprotokoll-Netzwerkimplementierung**.
- Das **Socket-Interface** als allgemeine Netzwerk-Programmierschnittstelle, das von vielen Systemen übernommen wurde.
- Die **virtuelle Speicherverwaltung** unter Unix wurde in Berkeley für die VAX 780 Systeme realisiert.
- Neue Kommandos: die C-Shell **csh**, der Editor **vi** sowie die remote Commands **rsh**, **rexec** und **rcp**.

Als **Berkeley System Distribution (BSD)** wurde es die Basis für eine Reihe von kommerziellen Systemen, auch für **MacOS**.

MINIX wurde von Andrew S. Tanenbaum an der Freien Universität Amsterdam als Lehrsystem entwickelt. Der Quellcode des Minix Kernels ist Teil von Tanenbaums Lehrbuch *Operating Systems – Design and Implementation*.

Das System wurde um 1987 zunächst auf auch für Studenten verfügbarer Hardware (PC mit Intel 8088 Prozessor, 512 Kilobyte RAM, ein Diskettenlaufwerk) entwickelt, enthielt aber alle Systemaufrufe der Unix-Version 7. Es verwirklichte Mehrprogrammbetrieb, Prozesse, Pipes, Signale und enthielt neben einem Mikrokernell die Neuimplementierungen vieler Unix Kommandos, einen Texteditor und einen C-Compiler. Aufgrund fehlender Hardware-Unterstützung war kein Speicherschutz und kein virtueller Speicher realisiert, auch die Netzwerkunterstützung fehlte zunächst. Später wurde das System auf andere Prozessoren (Intel 80286 und 80386, Motorola 68000-Linie, Sun SPARC) portiert und erweitert.

Minix diente dem finnischen Informatik-Studenten **Linus Torvalds** als Entwicklungsumgebung für seinen Kernel **Linux**.

AT&T war es nach dem Anti-Trust Verfahren erlaubt, in das Geschäft mit Computer einzusteigen und sie boten als Produkt das **Unix System V** an. Dieses wurde schnell die Basis für kommerzielle Produkte wie **HP-UX** von HP, **Solaris** von SUN (jetzt Oracle), **UTS** von Amdahl sowie **AIX** für S/370 und PowerPC von IBM.

Unix und Linux auf System z Teil 2

z/OS as a UNIX System

Unix

Welche Server-Betriebssysteme findet man in der Praxis ?

- Windows (verschiedene Varianten)
- Unix (verschiedene Varianten)
- i-Series (OS/400)
- zSeries Betriebssysteme (z/OS, zVM, zVSE, TPF)

Welche wesentlichen Unix-Varianten existieren ?

- HP/UX
- SunSolaris
- IBM AIX
- Siemens Sinix
- MacOS (BSD)
- Linux, einschließlich zLinux
- z/OS Unix System Services



Unix-Systeme sind weitestgehend, aber nicht 100 % kompatibel.

Was ist ein Unix-System ?

Ein Betriebssystem ist ein Unix-System, wenn es den Posix (1003.1 und 1003.2) Standard und die X/Open (XPG4 und XPG4.2) Standards erfüllt. Linux wurde nie Posix- und XPG4.2-zertifiziert, dürfte aber zu 99,9 % Posix- und XPG4.2-kompatibel sein. Ob Linux als Unix-System bezeichnet werden kann, ist umstritten.

Unix auf Mainframes

Es gab mehrere Versuche, Unix auf Mainframes und insbesondere auf System z und seine Vorgängern S/370 und S/390 ESA zu portieren und als Produkte anzubieten.

Der erste Ansatz kam von AT&T selbst, die für den Betrieb und den Abrechnungen ihres weltweites Telefonnetzwerk schon immer IBM Mainframes eingesetzt haben und nun versuchten, alle Anwendungen auf die einheitliche Softwareplatform UNIX zu portieren. Dazu benutzen sie das IBM Spezialsystem **Time Sharing System (TSS)** als **Hardware Abstraction Layer (HAL)**, um darauf Unix zu implementieren. Nachdem das Projekt innerhalb von AT&T erfolgreich realisiert wurde, versuchte IBM, dieses Konstrukt als virtuelles Betriebssystem unter VM unter dem Namen **AIX/ESA** als Produkt anzubieten. Aber durch die Kombination der verschiedenen Softwareebenen VM, TSS und Unix waren aber die Pfadlängen der Supervisor-Calls sehr lang und die Performance entsprechend miserabel. Außerdem widersprachen sich die Konzepte zwischen den S/370 Kanal-basierten 3270 Terminals und den raw-data Inputgeräten der Minicomputer, so dass kein effizienter Terminalbetrieb möglich war.

Die Firma Amdahl wählte für ihr Produkt **Universal Timesharing System (UTS)** einen anderen Ansatz und arbeitete ab 1975 mit der Princeton University zusammen, um Unix als virtuelles System direkt auf VM/370 zu implementieren. Dieses Produkt wurde 1980 für die Mainframe-kompatiblen Amdahl Computer angekündigt und erst 6 Jahre später gab es eine Version, die auch ohne VM direkt auf der Mainframe Hardware lief. Dieses System hatte nicht den Anspruch, full-screen Terminal I/O zu unterstützen, sondern bot nur ein Command-line Interface über die 3270 Schnittstellen an. Es lief seinerzeit auf etwa 300 Mainframes.

Ein weiteres Mainframe-Unix war **Hitachi HI-OSF/1-M**, das auf IBM kompatiblen Mainframe Rechner von Hitachi lief und eine weitere Variante von AT&T Unix System V.

z/OS vs. Unix

Führende Unix-Großrechner sind

- **Integrity Superdome** von HP mit Itanium-Prozessoren und dem **HP-UX** Betriebssystem
- **M9000** bzw. **SPARC M5-32 Server** von Oracle mit Sparc-Prozessoren und dem **Solaris** Betriebssystem.
Ein verwandtes Produkt wird von der Firma Fujitsu unter dem Namen „SPARC Enterprise Server“ vertrieben.
- **System p** von IBM mit PowerPC Prozessoren und dem **AIX** Betriebssystem

Neben den proprietären Unix-Dialektten ist auf diesen Rechnern auch Linux verfügbar.

Die I/O-Leistung eines Rechners wird gemessen in der Anzahl von I/O-Operationen pro Sekunde unter realistischen Betriebsbedingungen. Konkrete Untersuchungen sind nie veröffentlicht worden, aber es wird allgemein angenommen, dass die z/OS I/O-Leistung vielleicht um einen Faktor 3 - 10 höher als die I/O-Leistung von großen Unix-Rechnern wie Superdome und M9000 ist.

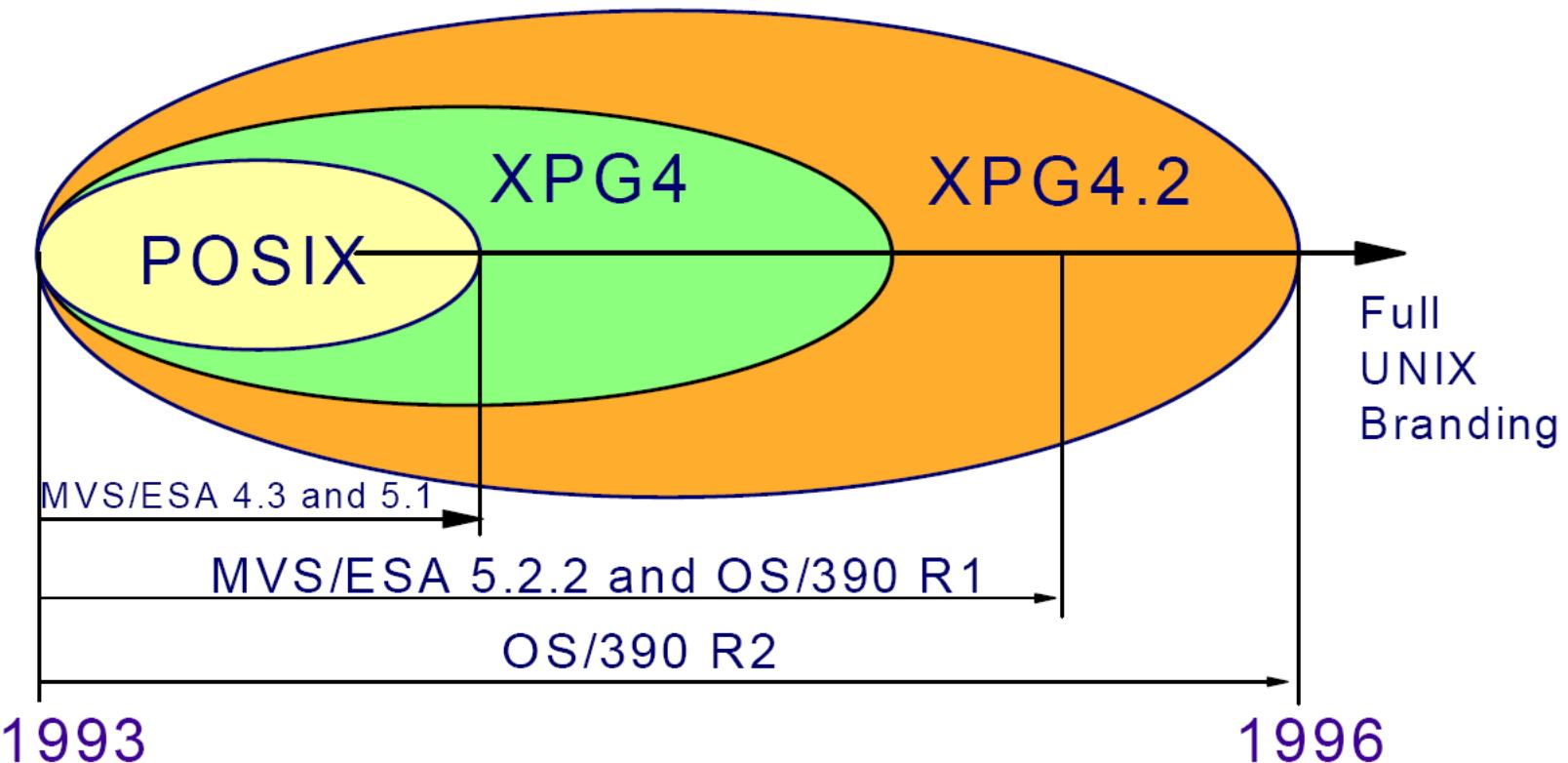
Was definiert ein Unix-System?

Unix wurde ursprünglich von Ken Thompson und Dennis Ritchie an den Bell Telephone Laboratories für Rechner der Digital Equipment Corporation (DEC) entwickelt und 1971 erstmalig im praktischen Betrieb eingesetzt. Schon bald entstanden (fast) kompatible Implementierungen für andere Rechner.

Aus Bemühungen um einen einheitlichen Unix Standard entstand das Portable Operating System Interface (POSIX). Dies ist ein gemeinsam von der IEEE und der Open Group für Unix entwickeltes, standardisiertes Application Programming Interface, welches die Schnittstelle zwischen Applikation und dem Betriebssystem darstellt.

X/Open ist eine Erweiterung des POSIX-Standards, welche ursprünglich von einem Konsortium mehrerer europäischer Hersteller von Unix-Systemen erarbeitet wurde. Die Erweiterung wurde in mehreren X/Open Portability Guides (XPG) veröffentlicht. Die letzten Versionen haben die Bezeichnungen XPG4 und XPG4.2.

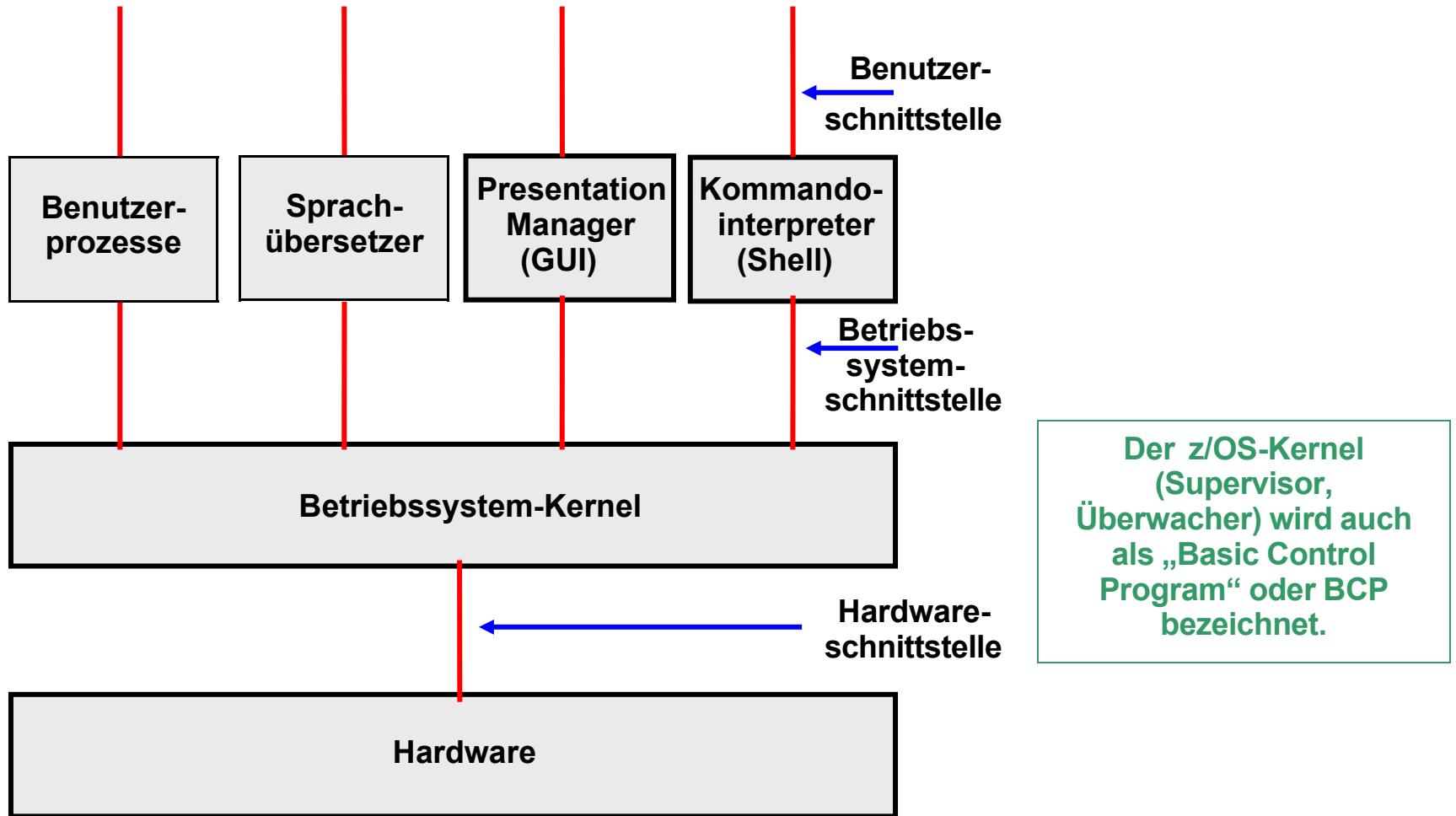
HP-UX (HP), Solaris (SUN/Oracle), AIX (IBM) und z/OS **Unix System Services** (IBM) sind heute die wichtigsten Unix-Betriebssysteme. Sie sind POSIX- und X/Open-zertifiziert. Dies garantiert, dass der Quellcode beliebiger Anwendungen mit minimalem Aufwand von einem Unix-System auf ein anderes Unix-System portiert werden kann, obwohl die Implementierungen sehr unterschiedlich sind.



z/OS „Unix System Services“ wurde 1993 erstmalig unter dem Namen Open MVS (OMVS) verfügbar. Es gilt als vollwertiges Unix-Betriebssystem.

Wie ist es möglich, dass z/OS gleichzeitig ein Unix-Betriebssystem sein kann?

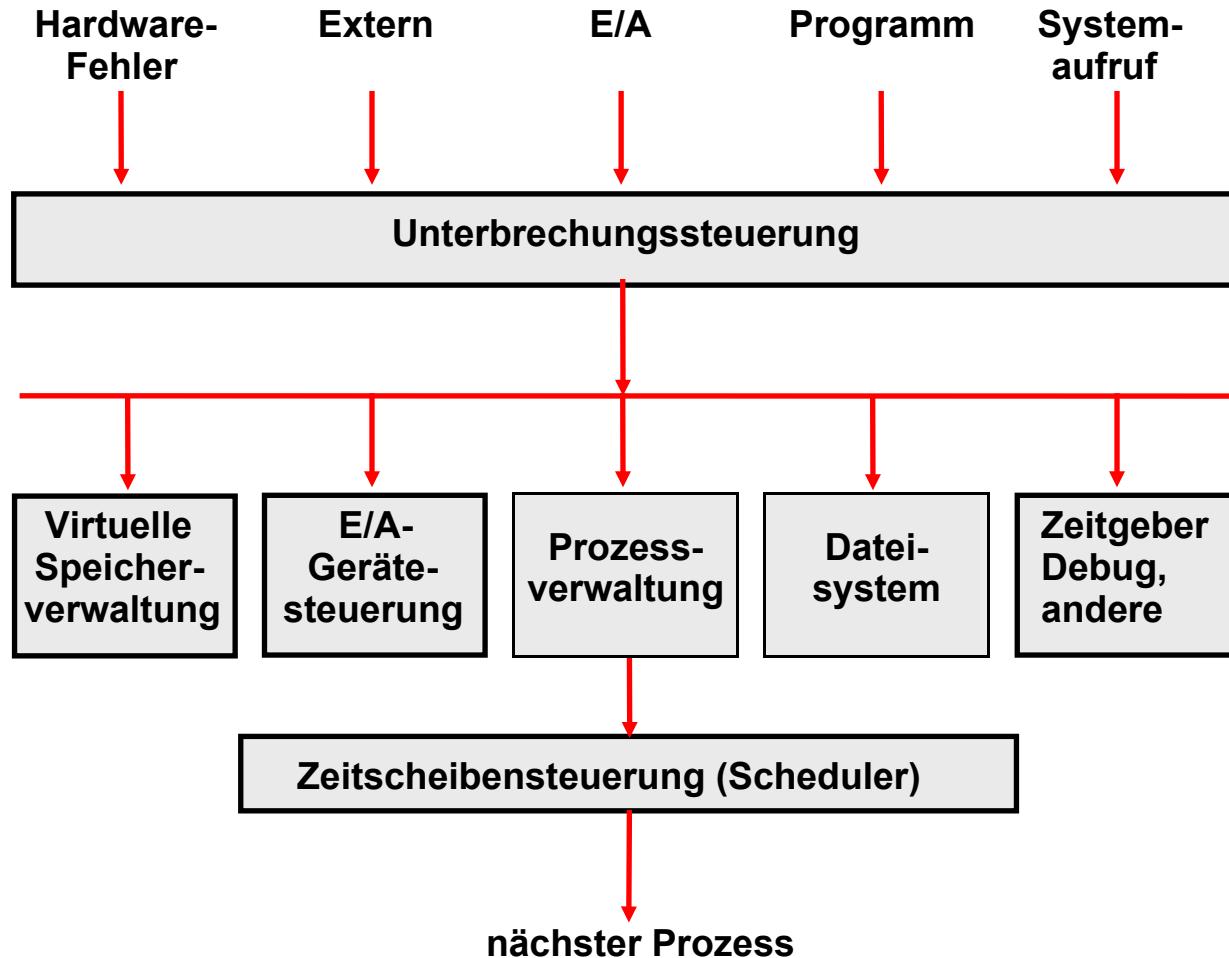
Schichtenmodell der Rechnerarchitektur



Betriebssystem-Kernel haben grundsätzlich immer eine sehr ähnliche Struktur. Sie bestehen aus den in der folgenden Abbildung dargestellten Komponenten.

Diese Folien sind teilweise eine Wiederholung von „Verarbeitungsgrundlagen Teil 3“ aus dem Vorlesungsskript „Einführung in z/OS“. Siehe <http://jedi.informatik.uni-leipzig.de/de/Vorles/Einfuehrung/Grundlag/vg03.pdf#page=08>

Struktur des Supervisors



Der Aufruf des Supervisor (Überwachers) eines Kernels erfolgt grundsätzlich über Unterbrechungen.

Je nach Art der Unterbrechung werden unterschiedliche Komponenten des Supervisor aufgerufen.

Benutzerprozesse nehmen Dienste des Supervisor über eine Schnittstelle, den Systemaufruf (System Call, Supervisor Call, SVC) in Anspruch. Diese Begriffe haben alle die gleiche Bedeutung.

Der Scheduler (Zeitscheibensteuerung) sucht den nächsten auszuführenden Prozess aus.

Supervisor Calls

z/OS SVC's (Supervisor Calls) sind das Äquivalent zu den Unix System Calls.

Supervisor Calls im weiteren Sinne sind Library-Routinen, die eine Dienstleistung des z/OS-Kernels in Anspruch nehmen.
Andere Bezeichnung: System Calls.

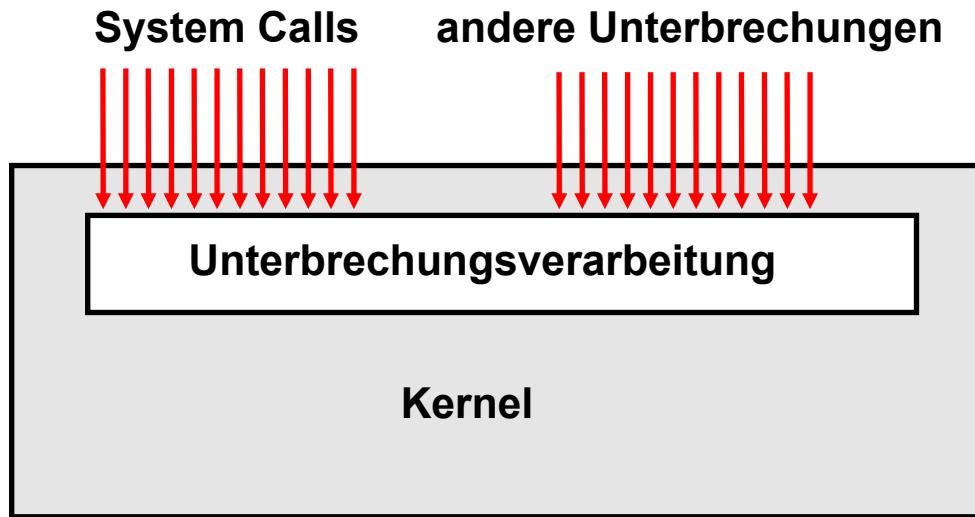
Supervisor Calls im engeren Sinne sind Maschinenbefehle, die mittels eines Übergangs vom User Mode (Problem Status) zum Kernel Mode (Supervisor Status) einen Interrupt Handler des Kernel aufrufen. Äquivalente Maschinenbefehle in der X86 (Pentium)-Architektur sind int 0x80 Call Gate und SYSENTER. Für die IA-64-Architektur wird die EPC (Enter Privileged Mode) instruction benutzt.

Ein SVC-Maschinenbefehl enthält einen 8 Bit Identifier, welcher die Art des Supervisor Calls identifiziert.

Beispiele sind:

GETMAIN	SVC 10	Anforderung von Virtual Storage
OPEN	SVC 19	Öffnen eines Data Sets
EXCP	SVC 0	Lesen oder Schreiben von Daten
WAIT	SVC 19	Warten auf ein Ereignis, z.B. Abschluss einer Leseoperation

Unix System Services sind eine Erweiterung des z/OS-Kernel (BCP) um 1100 Unix System Calls, die als z/OS SVCS implementiert sind.



Die Kernel aller Betriebssysteme haben de facto identische Funktionen, z. B.

- Unterbrechungsverarbeitung
- Prozessmanagement
- Scheduling / Dispatching
- Ein/Ausgabe-Steuerung
- Virtuelle Speicherverwaltung
- Dateimanagement

Ein Unix-Kernel unterscheidet sich von einem Windows-Kernel durch die Syntax und Semantik der unterstützten System Calls, seine Shells sowie durch die unterstützten Dateisysteme.

Linux, Solaris, HP-UX und AIX haben unterschiedliche Kernel, aber (nahezu) identische System Calls..

Der Kernel eines Betriebssystems wird fast ausschließlich über Unterbrechungen oder über System Calls aufgerufen.

Unix System Services (USS)

Was ist ein Unix-Betriebssystem?

Ein Unix-Betriebssystem erfüllt den „POSIX“-Standard. Das „Portable Operating System Interface“ ist ein gemeinsam von der IEEE und der Open Group für Unix entwickeltes, standardisiertes Application Programming Interface, das die Schnittstelle zwischen Applikation und dem Betriebssystem darstellt. Der Standard beinhaltet außerdem Spezifikationen für ein Command Line Interface, Services wie „basic I/O“ (File, Terminal und Netzwerk), sowie eine threading Library API.

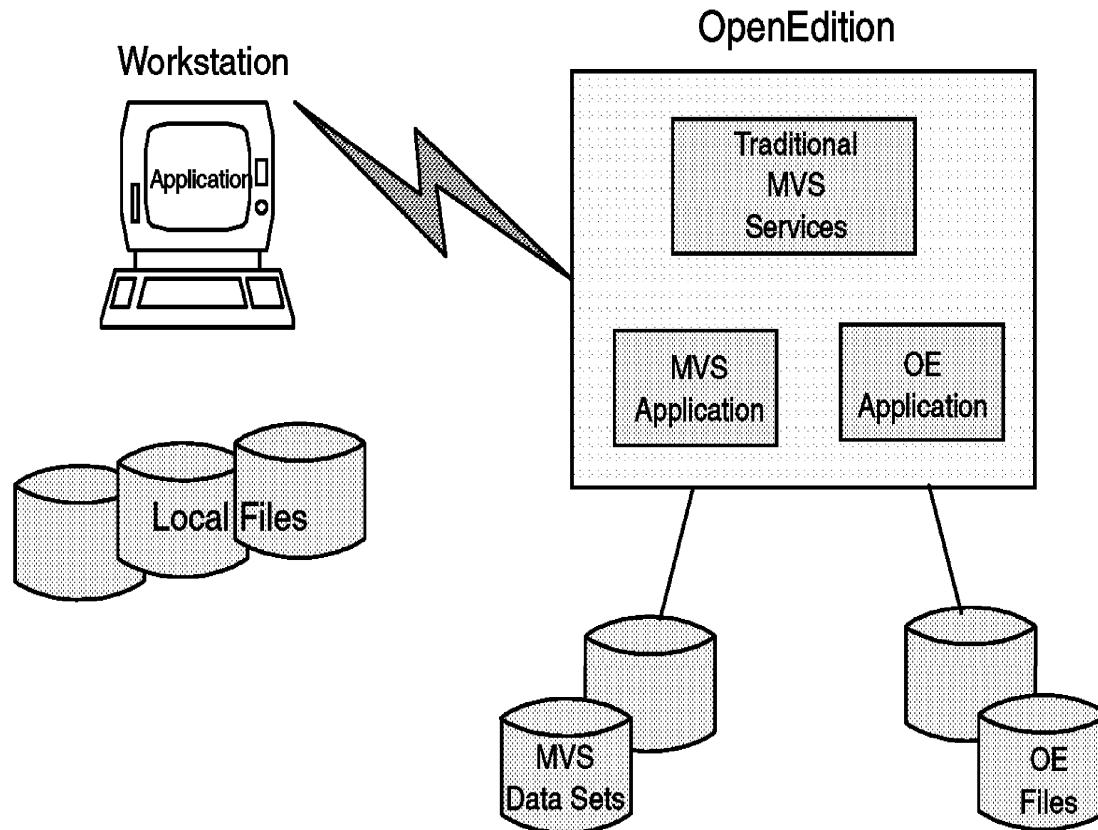
Die wichtigsten Implementierungen sind AIX, HP-UX, Solaris und Mac OS X.

Die Unix System Services (USS) des z/OS-Betriebssystems sind eine Erweiterung des z/OS-Kernel um 1100 Unix System Calls, zwei Unix Shells und zwei Unix-Dateisysteme. Sie erfüllen den POSIX-Standard.

Damit wird aus z/OS ein Unix-Betriebssystem. Dies ermöglicht eine einfache Portierung von Unix-Anwendungen nach z/OS. Ein typisches Beispiel ist das SAP-R/3-System, welches ursprünglich für das Unix-Betriebssystem entwickelt wurde, aber auch unter „z/OS Unix System Services“ lauffähig ist.

z/OS Unix System Services wurde früher als „Open Edition MVS“ bezeichnet.

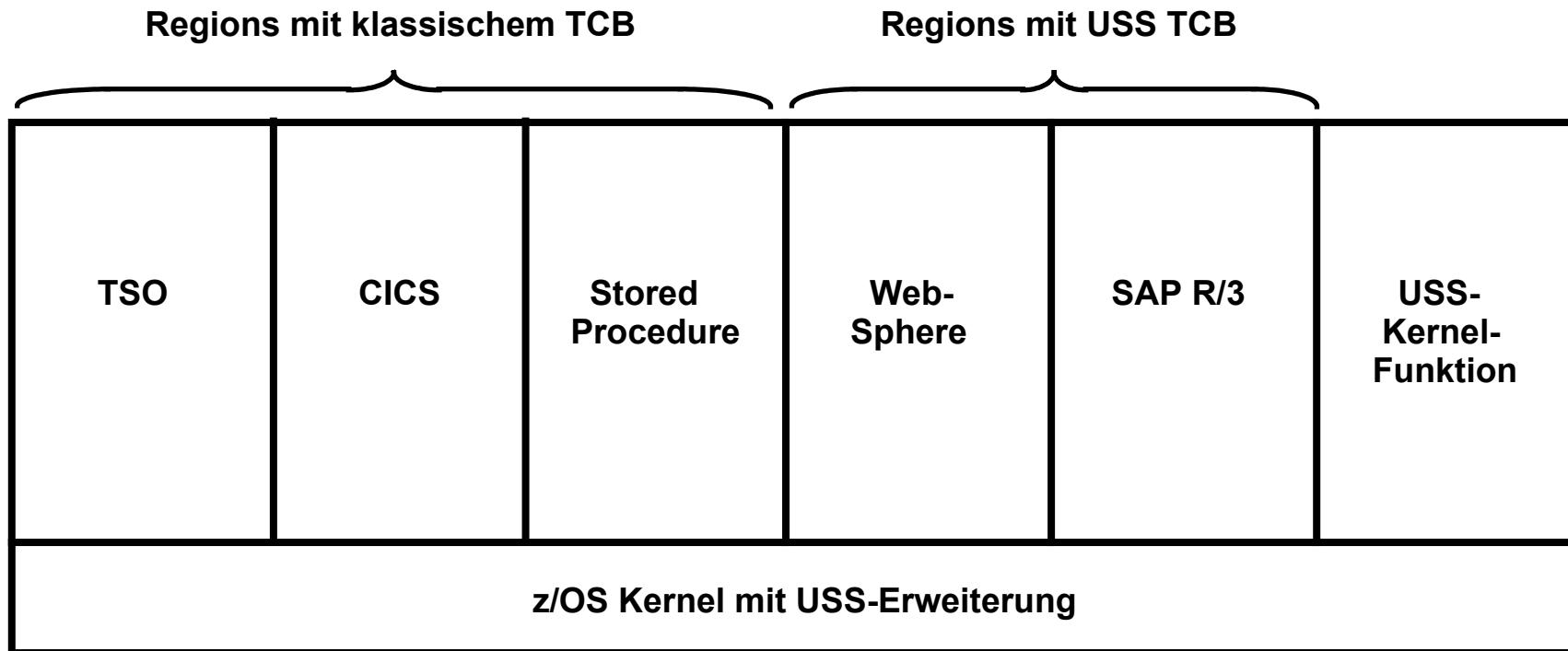
Unix System Services Open Edition



z/OS verhält sich entweder wie ein traditionelles MVS-System (z/OS) oder wie ein Unix-System.

z/OS Unix System Services wurde früher als Open Edition (OE) oder Open MVS bezeichnet.

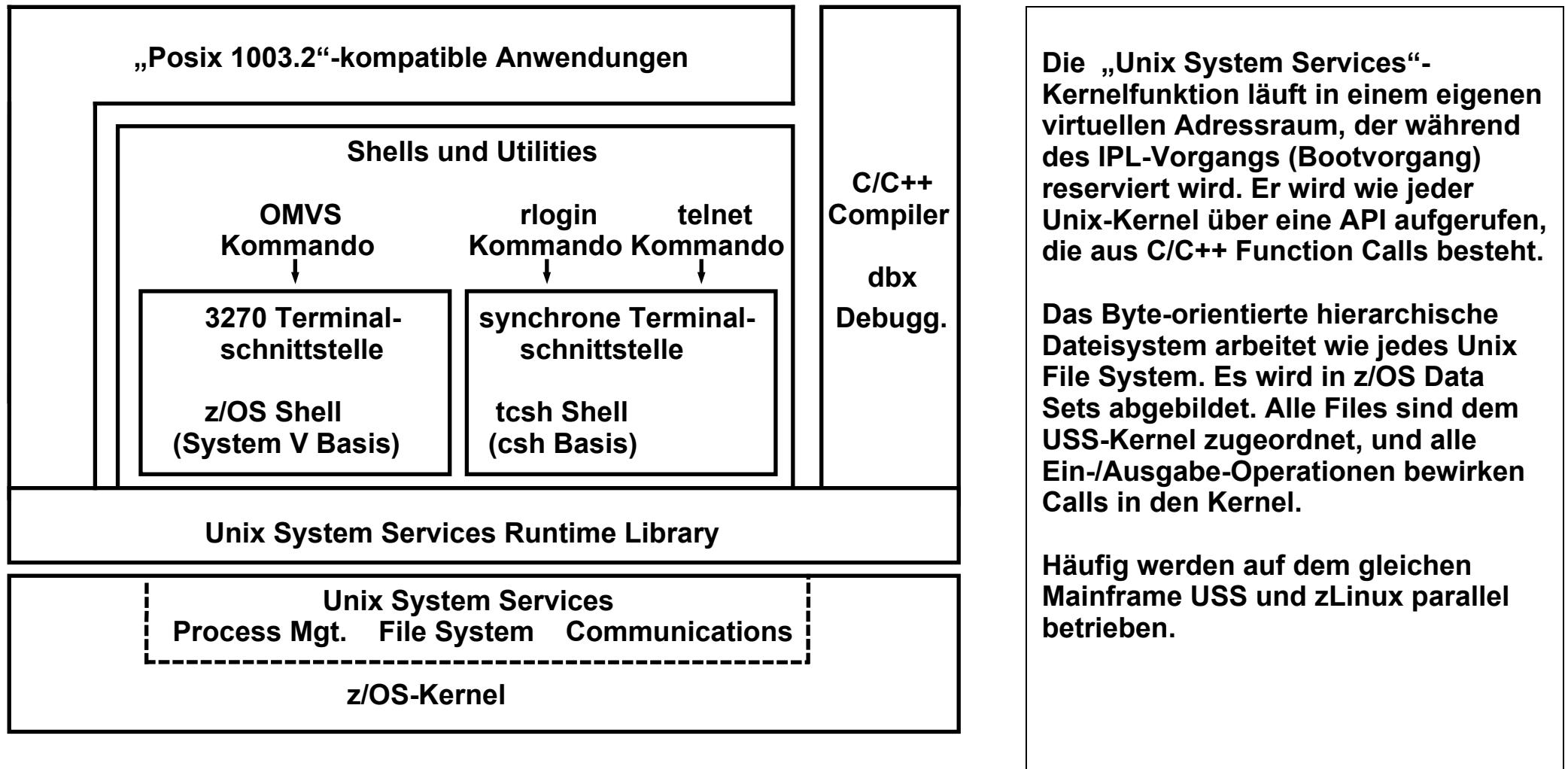
Erweiterter Task Control Block (TCB)



Regions, welche Unix System Services einsetzen, benutzen einen modifizierten (erweiterten) TCB.

Die USS-Kernel-Funktionen laufen in einem eigenen virtuellen Adressraum, der während des IPL-Vorgangs (Bootvorgangs) reserviert wird.

Unix System Services (USS)



z/OS UNIX-Komponenten

Ein typisches Unix-Betriebssystem besteht aus einem Kernel, der eine direkte Schnittstelle mit der Hardware darstellt. Aufgebaut auf dem Kernel ist die Shell und eine Utilities-Schicht, die ein Kommando-Interface definiert. Dazu gibt es Anwendungsprogramme, die auf der Shell oder auf Utilities aufgebaut sind.

Beim z/OS-Systemstart (IPL) werden USS-Kernel-Dienste automatisch gestartet. Der Kernel bietet z/OS USS als Service für Anforderungen von Unix-Programmen und von der USS Shell an. Der Kernel verwaltet ein Unix-Dateisystem (HFS), die Kommunikationseinrichtungen und die USS-Prozesse. Das hierarchische Filesystem wird als Teil des Kernels betrachtet, und als Komponente des DFSMS (Data Facility Storage Management Subsystem) installiert.

Siehe <http://jedi.informatik.uni-leipzig.de/de/Vorles/Einfuehrung/Bs/zos03.pdf#page=016>.

Auch Nicht-Unix-Programme können auf das hierarchische Filesystem zugreifen. So kann z.B. ein CICS-Programm im Zusammenhang mit CICS Web Support auf HFS zugreifen.

Um diese APIs zu unterstützen, muss das z/OS-System einige Systemdienste in seinem Kernel enthalten, z. B. die hierarchischen Filesystem- und Kommunikationsdienstleistungen.

Daemons sind Programme, die typischerweise gestartet werden, wenn das Betriebssystem initialisiert wird. Sie bleiben aktiv, um standard Services auszuführen. z/OS USS enthält Daemons, die auf Anforderung Anwendungen oder eine User-Shell-Sitzung starten.

Unix System Services Komponenten

z/OS UNIX Shell und Utilities

Ein interaktives Interface zu z/OS Unix Services, welche Kommandos von interaktiven Benutzern und von Programmen interpretiert.

Die Shell- und Utilities-Komponente kann mit den TSO-Funktionen unter z/OS verglichen werden.

Hierarchical File System

Neben den vorhandenen z/OS-Dateisystemen (z.B. VSAM, PDS) wurde ein zusätzliches Unix-kompatibles hierarchisches Dateisystem eingerichtet. Jedes Programm kann auf das hierarchische Dateisystem zugreifen.

z/OS UNIX Debugger (dbx)

Der z/OS-Unix-Debugger ist ein Werkzeug, das Anwendungsprogrammierer verwenden, um interaktiv ein C-Programm zu debuggen. Der dbx-Debugger ist nicht Teil des POSIX-Standards. Es basiert auf dem Unix-dbx-Debugger, der auch in vielen Unix-Umgebungen vorhanden ist.

C/C++ Compiler und C run-time Libraries

Die C/C++-Compiler und C-Laufzeitbibliotheken werden benötigt, um ausführbare Dateien zu erstellen, die der Kernel verstehen und verwalten kann. Die C/C++-Compiler und „Language Environment (LE) Feature“-Bibliothek wurden verändert und erweitert, um Unterstützung für die POSIX- und XPG4-C-Funktionsaufrufe zu enthalten. Das LE-Produkt stellt eine gemeinsame Laufzeitumgebung und sprachspezifische run-time Services für kompilierte Programme zur Verfügung.

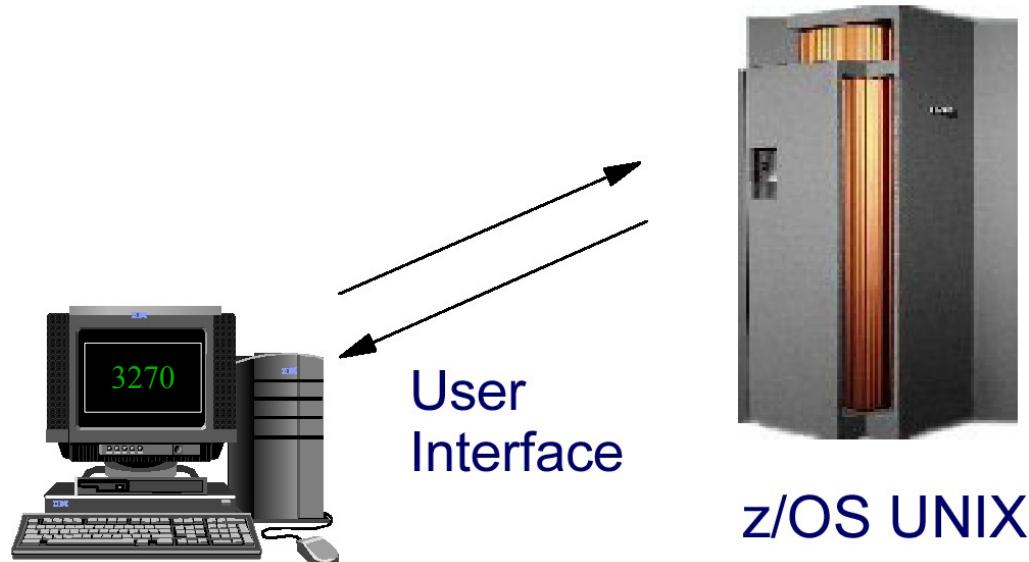
Um einen Shell-Befehl, ein Dienstprogramm oder ein vom Benutzer bereitgestelltes Anwendungsprogramm in C oder C++ auszuführen, benötigen Sie die „C/C++ Runtime Library“, die mit LE zur Verfügung gestellt wird.

TSO und UNIX System Services

Abhängig von der Anwendung gibt es 3 Modi, in denen die Benutzerschnittstelle operieren kann:

- **line mode**
 - der Input wird nach Drücken der <Enter> Taste verarbeitet
- **raw mode**
 - jedes Zeichen wird sofort nach der Eingabe auf der Tastatur verarbeitet
- **graphical mode**
 - graphische Benutzerschnittstelle für X-Window Anwendungen

Es existieren zwei alternative Möglichkeiten für den Zugriff auf Unix System Services.



Eine Möglichkeit ist die Benutzung der z/OS-Shell mittels eines 3270-Terminals (oder einer 3270 Emulation) und TSO. Die z/OS-Shell gleicht der „Unix System V“-Shell mit einigen zusätzlichen Eigenschaften der Korn-Shell. Sie benutzt den ISPF-Editor. Ein Benutzer startet eine TSO Session und gibt das OMVS-Kommando ein. Es ist nur **line mode** möglich. Dies ist die bevorzugte Methode für TSO-Experten.

Menu List Mode Functions Utilities Help

ISPF Command Shell

Enter TSO or Workstation commands below:

==> omvs 

Place cursor on choice and press enter to Retrieve command

=> ish
=> omvs
=>
=>
=>
=>
=>
=>
=>
=>

F1=Help F2=Split F3=Exit F7=Backward F8=Forward F9=Swap
F10=Actions F12=Cancel

Die Unix System Services Shell wird vom ISPF Command Shell Screen durch die Eingabe des TSO-Kommandos "OMVS" gestartet.

(C) Copyright Software Development Group, University of Waterloo, 1989.

All Rights Reserved.

U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or Disclosure restricted by GSA-ADP schedule contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.

```
SPRUTH : /u/spruth >ls -als
total 96
16 drwxr-xr-x    4 BPXOINIT SYS1          8192 Oct  3 23:40 .
16 drwxrwxrwx   120 BPXOINIT SYS1         8192 Sep 30 17:32 ..
8 -rwx-----     1 BPXOINIT SYS1         365 Mar 25 2002 .profile
8 -rw-----     1 BPXOINIT SYS1        2347 Oct  3 23:42 .sh_history
8 -rw-r-----    1 BPXOINIT SYS1        3715 Jul  7 2001 index.htm
8 -rw-r-----    1 BPXOINIT SYS1        2806 Jul  7 2001 links01.htm
16 drwxr-x---    2 BPXOINIT SYS1        8192 Mar 28 2002 sm390
16 drwxr-xr-x    6 BPXOINIT SYS1        8192 Apr 12 20:06 was_samples
```

SPRUTH : /u/spruth >

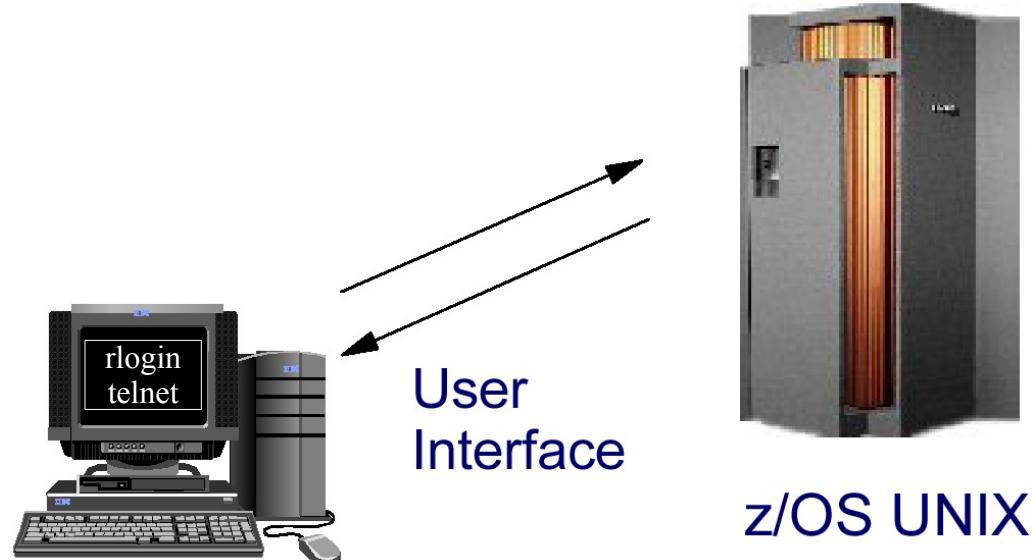
==>

INPUT

ESC=¢	1=Help	2=SubCmd	3=HlpRetrn	4=Top	5=Bottom	6=TSO
	7=BackScr	8=Scroll	9=NextSess	10=Refresh	11=FwdRetr	12=Retrieve

In dem gezeigten z/OS-Shell-Beispiel hat der Benutzer /u/spruth den Unixbefehl ls -als eingegeben

Die andere Möglichkeit ist das **asynchrone Terminal Interface**:



Sie wird über rlogin oder telnet (z.B. mittels putty) aufgerufen. Beide Modi bieten sowohl **line mode** als auch **raw mode** an. D.h. der Benutzer kann zum Beispiel Anwendungen wie den vi-Editor verwenden. Als Shells kann der Benutzer die standard OS/390 Shell verwenden oder die tcsh-shell, eine modifizierte C-Shell Variante.
Es ist weiterhin möglich, graphische **X-Window** Anwendungen zu starten, vorausgesetzt, die Workstation ist als X-Server konfiguriert.

Dies ist die bevorzugte Methode für Unix-Experten.

Benutzung von z/OS UNIX System Services

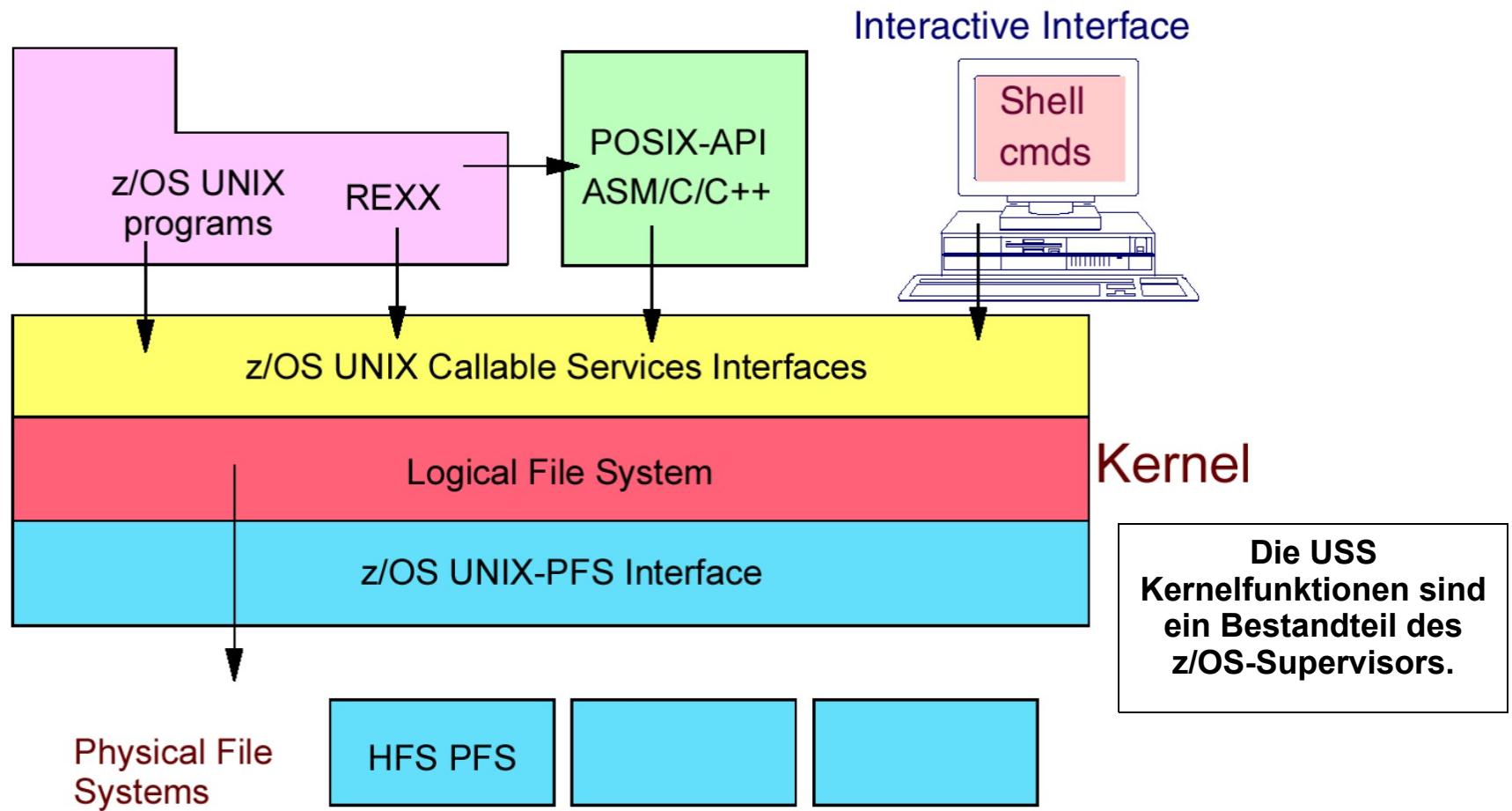
Ein Benutzer kann mit z/OS Unix über die folgenden Schnittstellen interagieren:

- Das Application Programming Interface (API) besteht aus Aufrufen (Calls), die von C/C++-Programmen verwendet werden können, um auf z/OS-Unix zuzugreifen. Diese C-Aufrufe sind in dem „POSIX 1003.1“-Standard definiert.

Die aufrufbaren Dienstleistungen können direkt von Assembler-Programmen verwendet werden, um auf z/OS-Unix zuzugreifen, z. B. auf Dateien in dem hierarchischen Dateisystem. Diese Möglichkeit erlaubt es anderen Hochsprachen und Assembler, z/OS-Unix zu verwenden. Die API-Schnittstelle bietet die Möglichkeit, XPG4.2-Programme auf z/OS auszuführen. Ein Programm, das dem XPG4.2-Standard entspricht, kann auf einem System entwickelt und dann auf ein anderes System portiert werden. Dort kann es dann kompiliert, gelinkt und ausgeführt werden. Ein solches Programm wird als portable (portierbar) bezeichnet.

- Die interaktive Schnittstelle wird als z/OS-Unix-Shell bezeichnet. Die Shell ist ein Kommando-Interpreter, der Befehle akzeptiert, die in dem „POSIX 1003.2“-Standard definiert sind. Shell-Befehle können in einer Datei als ein Shell-Skript gespeichert und dann ausgeführt werden. Das Shell-Skript arbeitet ähnlich wie z/OS CLISTS oder REXX EXECs.

TSO REXX enthält Erweiterungen, um den Zugang zu den z/OS-Unix-abrufbaren Dienstleistungen zu ermöglichen. Eine REXX EXEC mit USS kann von TSO/E ausgeführt werden, als z/OS Batch Job oder in der Shell.



Mit beiden Shells kann ein standard Unix **Hierarchical Filesystem (HFS)** benutzt werden. z/OS verfügt über ein logisches hierarchisches Dateisystem, welches auf eins von mehreren verfügbaren physischen Dateisystemen abgebildet wird. So wie Linux über mehrere physische Dateisysteme verfügt (ext2, ext3, ext4, ReiserFS), existieren auch für z/OS USS mehrere physische Dateisysteme.

Unix System Services versus zLinux

Für die heutigen Mainframes sind aktuell zwei Unix-Implementierungen von Bedeutung:

- zLinux
- z/OS Unix System Services (USS)

Sie sind vor allem von Interesse, wenn existierende Unix-/Linux-Anwendungen von einer distributed platform auf den Mainframe portiert werden sollen.

Warum zwei Alternativen und was ist der Unterschied?

zLinux ist ein regulärer Port von Suse oder Red Hat Linux auf System z Hardware. Es läuft entweder in einer eigenen LPAR, oder als virtuelle Maschine unter z/VM. Beide Alternativen sind auf Mainframe-Servern relativ weit verbreitet und werden fast immer nicht an Stelle von z/OS, sondern parallel zu z/OS benutzt.

Auf der Betriebssystem-Ebene ist der zLinux-Kernel performanter als der z/OS-Kernel. Es fehlen aber alle z/OS-spezifischen Funktionen, beispielsweise Unterstützung für Sysplex, Coupling Facility, Work Load Manager, RACF, FICON, Protection Keys, Krypto und vieles mehr. Je nachdem ob diese Funktionen gebraucht werden, laufen Unix-Programme entweder unter zLinux oder USS. Häufig wird in einer Mainframe-Installation beides genutzt.

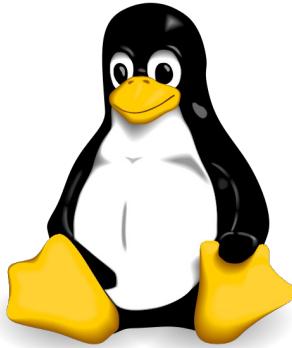
Im Gegensatz zu zLinux ist Unix System Services (USS) kein eigenes Betriebssystem, sondern ein integrierter Bestandteil von z/OS. Es ist damit in der Lage, alle z/OS-Funktionen zu nutzen. Es ermöglicht eine relativ problemlose Portierung von existierenden Unix-Anwendungen auf einen Mainframe-Server, ohne dabei auf existierende z/OS-Eigenschaften verzichten zu müssen.

Zwei der wichtigsten Softwarepakete, die unter z/OS Unix System Services laufen (oder auch unter zLinux), sind:

- SAP/R3
- WebSphere

Unix und Linux auf System z Teil 3

Linux



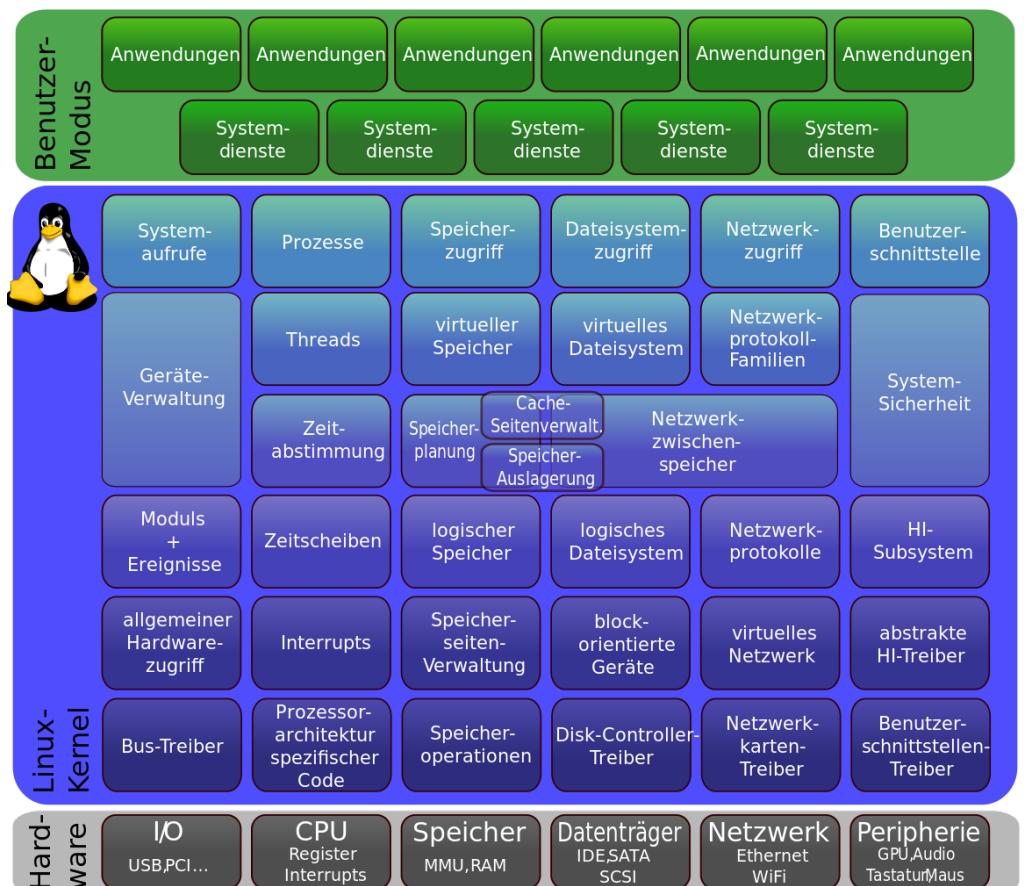
Entstehung von Linux

Linux wurde ca. 1991 als Projekt des finnischen Studenten **Linus Torvalds** begonnen, um einen frei verfügbaren Betriebssystem Kernel für die Intel Prozessoren 80386 zu entwickeln. Er benutze dazu **Minix** als Entwicklungsumgebung und als Compiler den **GNU C Compiler**.

Torvalds wollte zunächst die Fähigkeiten der neuen Intel 80386-Prozessorlinie (Multitasking, Paging) ausprobieren, entwickelte aber dann einen voll funktionsfähigen Kernel mit virtuellem Speicher und Speicherschutzmechanismen. Anders als bei seinem Vorbild, das einen **Microkernel** als Basis hatte, entschied er sich für einen **monolythischen Kernel**.

Der neue Kernel wurde erst unter einer privaten Lizenz veröffentlicht, da es Restriktionen für die kommerzielle Nutzung hatte. Da das Betriebssystem allein keinen Wert hatte und nur in Verbindung mit einer Reihe von Software Tools (bash shell, Compiler, etc.) benutzbar war, die in der Regel unter GNU Lizenzen verfügbar waren, entschied sich Torvalds 1992, den Kernel unter der **GNU General Purpose License** zu veröffentlichen.

Seit der Zeit ist Linux kontinuierlich weiterentwickelt worden und läuft auf einer großen Anzahl von Hardware-Architekturen und Systemen von embedded Realtime Controller bis zu Mainframe Systemen und Supercomputer. Linus Torvalds hat bis heute eine kontrollierende Rolle bei der Weiterentwicklung des Kernels und wird auch als [Benevolent Dictator for Life](#) bezeichnet.





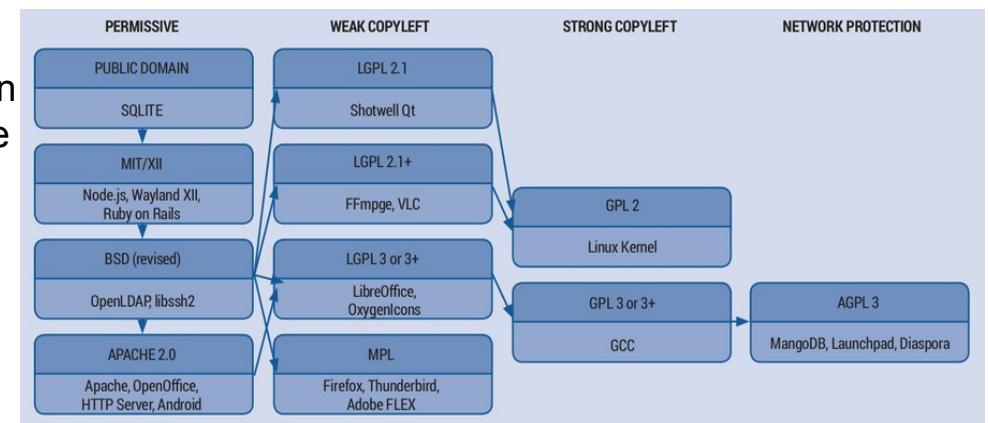
OpenSource Entwicklungen

In den Anfängen der Computerzeit wurde die Software zum Betrieb eines Computers vom Hersteller kostenlos mitgeliefert (meistens in Form von Source Code). Diese konnte dann von den Kunden verändert werden, z.B. um andere Geräte anzuschließen. Ab 1970 entstand eine Software Industrie, die vom Verkauf ihrer Produkte lebte. 1980 wurde das Copyright Recht in den USA erweitert, so dass es auch für Software galt.

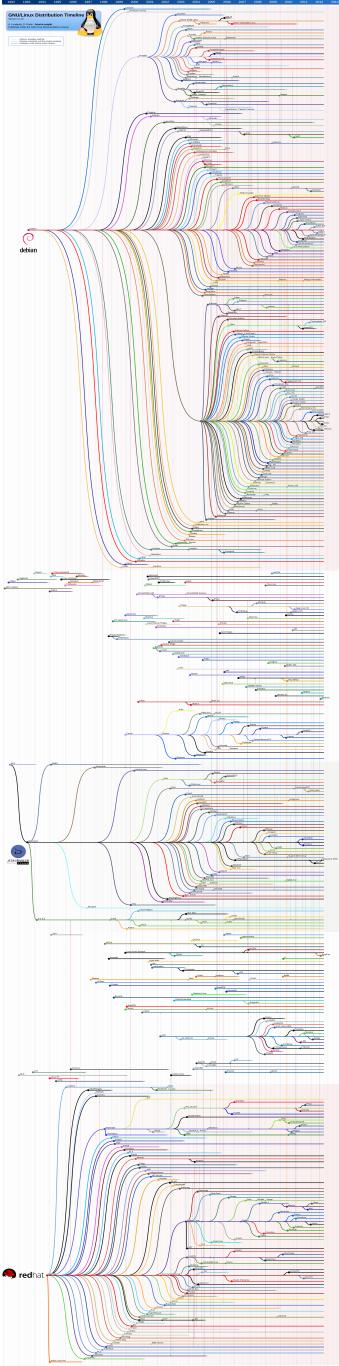
In 1983 wurde das **GNU Projekt** von **Richard Stallman** angekündigt mit dem Ziel, Software zu entwickeln und bereitzustellen, die dem Computerbenutzer erlaubt, sie zu benutzen, sie weiterzuentwickeln und weiterzugeben. GNU Software erlaubt diese Rechte legalisiert durch entsprechende Lizenzen und stellt damit **Free Software** her.

Es gibt verschiedene Varianten von Free Software, die festlegen, was man mit der Software anstellen kann:

- **Permissive** bedeutet, dass es (so gut wie) keine Restriktionen bezüglich der Verwendung und der Veränderung der Software gibt. Das bedeutet insbesondere auch, dass die Software proprietär werden kann
- **Copyleft** gibt dem Benutzer das Recht, die Software verändert oder unverändert weiterzuverteilen, solange dafür die gleichen Rechte gelten wie für die ursprüngliche Version
- **Strongly Protective** oder **Network Protection** verhindert, dass die Software proprietär werden kann



Am Anfang hatte die Open Source Bewegung eine Reihe von Compiler, Editoren, Werkzeugen und Anwendungen in ihrem Portfolio, aber keine eigene Platform. Durch den Linux Kernel von Torvalds änderte sich das und die Open Source Foundation konnte ein vollständiges System anbieten.



OpenSource Distribution

Komplette Linux Systeme werden in Form von Distributions dem Kunden angeboten. Deren Ziele sind:

- Zuverlässig - das System ist getestet für eine bestimmte Zielplattform
- Stabil - es läuft auf den unterstützten Zielsystemen zuverlässig und stabil
- Effizient - es werden für die unterschiedlichen Zielsysteme Optimierungen durchgeführt
- Sicher - alle sicherheitsrelevanten Funktionen sind aktiv und werden regelmäßig erneuert

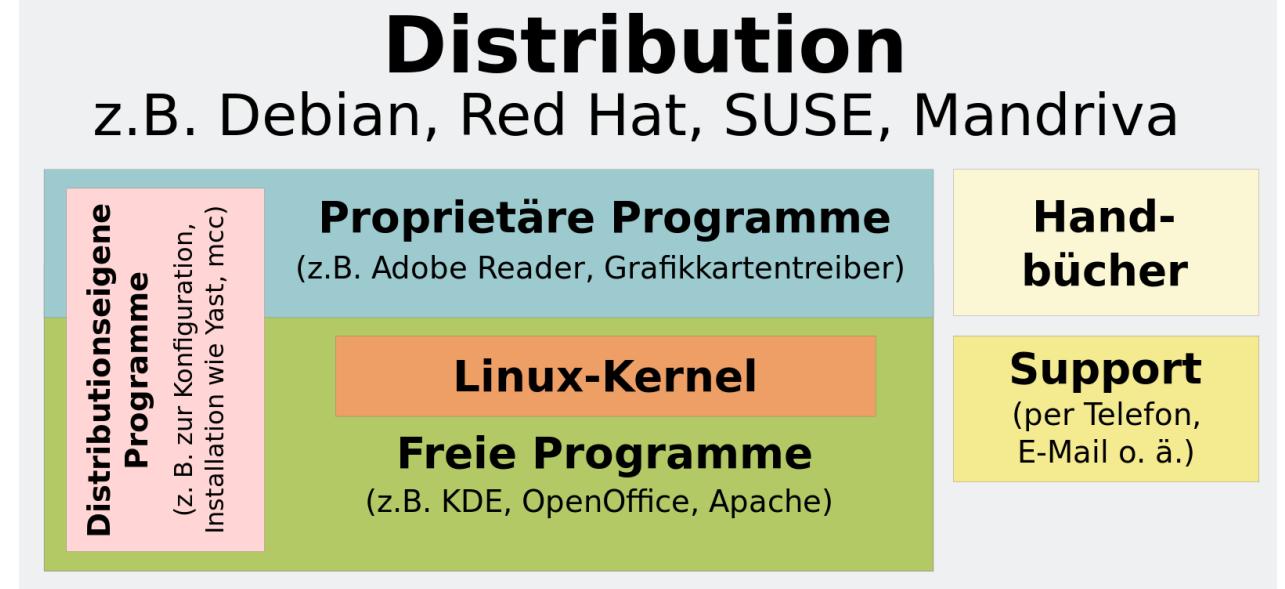
Mit anderen Worten, mit einer Distribution bekommt der Kunde ein lauffähiges System mit allen notwendigen Komponenten, die aufeinander abgestimmt und miteinander getestet sind.

In der Regel unterscheiden die Hersteller von Distributions zwischen kostenlosen **Private Editions** und kostenpflichtigen **Enterprise Editions**. Letztere haben meistens einen großen Funktionsumfang und werden von den Anbietern auch mit Service und Updates ausgestattet.

Die maßgeblichen Distributions für PCs, Workstations und Server sind:

- Ubuntu
- Debian
- RedHat
- SuSe

[Zeitleiste mit der Entwicklung verschiedener Linux-Distributionen](#)



Unix und Linux auf System z Teil 4

zLinux

History of Linux for System z

Ende 1990 begann ein Gruppe von jungen Software Entwicklern im IBM Entwicklungslabor in Böblingen in Form eines “**Skunk Projektes**”, das OpenSource Betriebssystem Linux auf einem Mainframe laufen zu lassen.

Das Ziel war, eine vollfertige Version von Linux auf die reine S/390 Architektur zu portieren.

Das bedeutet, dass

- Linux direkt (native) auf der S/390 Hardware laufen sollte
- Linux Block Devices mittels Channel Control Words (CCWs) Device Driver auf S/390 Platten (DASDs) zugreifen
- als Zeichenkodierung ASCII statt EBCDIC benutzt wurde und die S/390 Architektur dahingehend erweitert wurde, ASCII kodierte Zeichen zu verarbeiten.

Parallel begann 1998 ein Projekt “**Linux on VM port**” am Marist College, NY.

Eine Anzahl von IBM Mitarbeiter aus den USA beteiligten sich aus persönlichem Interesse, aber ohne offiziellen Auftrag.

Es wurde die Frage diskutiert, ob andere OpenSource Versionen wie BSD portiert werden sollte, aber man war der Meinung, dass BSD

Portierung mehr in Richtung PCs zielten, während Linux für eine breitere Bereich von Hardware Platformen ausgelegt war.

Dieses Projekt wurde später umbenannt in “**Bigfoot Linux**”.

Am 18. Dezember 1999 hat IBM offiziell Linux für S390 angekündigt:

“*Linux for S/390 will run under VM, in an LPAR and 'on the bare metal'*”

Außerdem hat IBM seine Source Code Modifikationen für den Linux Kernel und der dazugehörigen Tools Chain freigegeben.

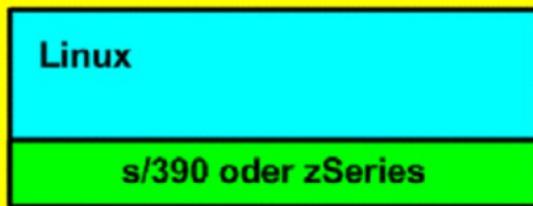
Was ist anders bei Linux auf S/390?

- kein VGA/SVGA monitor oder Graphic Karte
- keine Konsole Tastatur, keine Sound Karte, etc.
- keine Disketten- und CD Laufwerke

S/390 specific code			
	Total Lines of Code	S/390 Enabling Code	Percentage of S/390 Code
Linux	2,200,00	45,000	2.00
gcc	1,700,000	9,000	0.50
gdb	1,500,000	8,000	0.50
glibc	1,200,000	5,000	0.40
binutils	800,000	6,000	0.75
strace	41,000	200	0.50

zLinux auf dem Mainframe

a) Hardware native



b) Partitions native



c) z/VM native



d) Partitions und z/VM mixed

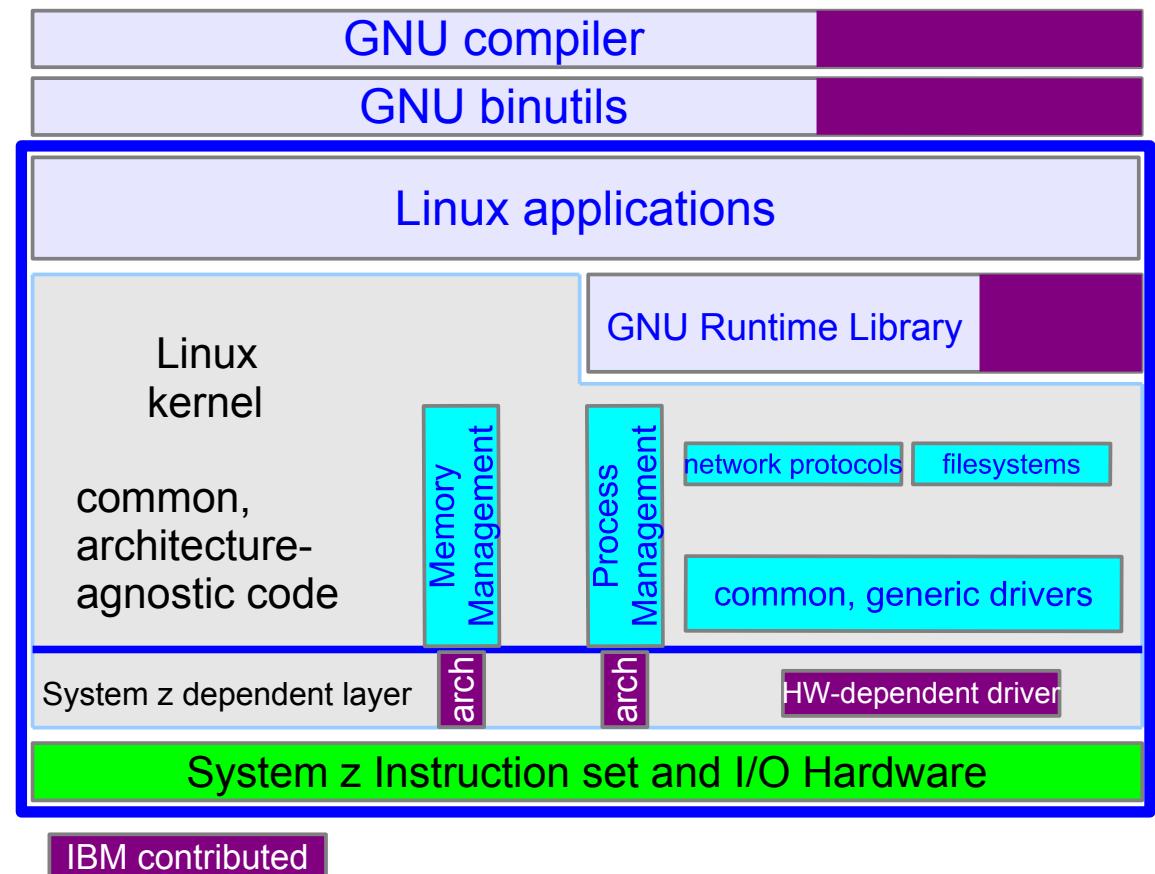


zLinux ist ein populäres Mainframe Betriebssystem. Es wird aber fast immer zusätzlich zu z/OS eingesetzt, und zwar entweder in einer z/VM virtuellen Maschine, oder in einer eigenen LPAR, oder beides. LPARs können parallel zu virtuellen Maschinen unter z/VM eingesetzt werden. Dargestellt sind unterschiedliche Alternativen.

Die Möglichkeiten a) und c) werden in der Praxis selten eingesetzt. Am häufigsten findet man die Varianten b) und d).

Linux Implementierung auf System z

- der einzige Unterschied zwischen zLinux und Linux für andere Platformen sind Teile des Compilers, der Runtime Libraries und einer hardware-abhängigen Komponente des Kernels
- 2 Versionen:
 - Linux for S/390: 32-bit
 - Linux for System z: 64-bit
- Linux auf System z ist vollständig OpenSource unter der GNU General Public License
- bekommt eine große Anzahl von Anwendungen aus der OpenSource Community und von ISVs, die ihre Linux Anwendungen auch auf zLinux portieren
- ermöglicht Server Konsolidierung, indem viele unabhängige Server Systeme auf einer einzigen Box zusammengefaßt werden
- eine große Anzahl von gut ausgebildeten Programmierer mit Linux Skill ist weltweit verfügbar



Distributoren von zLinux

Der Änderungen am Linux Kernel und an den notwendigen Entwicklungswerkzeugen (Tools-Chain mit Compiler, Loader, Utilities, etc.) für System z werden von IBM Entwicklern durchgeführt und gewartet . Dieser System z spezifische Code wird als zusätzliche Hardware Platform in die entsprechenden Versions Bäume released. Damit ist es OpenSource Code wie für jede andere Hardware Architektur auch und jede Person kann sich den Source Code runterladen, ein zLinux bauen und auf seinem Mainframe laufen lassen.

zLinux wird nicht von IBM als Produkt angeboten, sondern nur über Distributoren an die Kunden ausgeliefert.
Es gibt eine Vielzahl von Anbietern von zLinux auf dem Markt, z.B:

weltweite Anbieter wie **Caldera, Red Hat, SuSE, Turbolinux** sowie
lokale Anbieter **Conectiva, ESware, Mandrake, Red Flag, ...**)

Die am weitesten verbreiteten Versionen von zLinux sind

- **SUSE Linux Enterprise Server**
- **Red Hat Enterprise Linux**

Zusätzlich gibt es über 400 unabhängige Software Anbieter (**Independent Software Vendors ISVs**), die mehr als 1600 Anwendungen für Linux auf System z anbieten.

Auch IBM bietet viele seiner Middleware Produkte für zLinux an:

- **WebSphere** – Integration von Transaktionsverarbeitung
- **DB2** – Datenbanksystem
- **Tivoli** – System Management
- **MQ-Series** – Transaktionsbasiertes Kommunikationssystem

Was System z für Linux bringt:

die zuverlässigste Hardware Platform:

- redundante Processoren and Speicher
- Error detection and correction
- Remote Support Facility (RSF)

Zentralisierte Linux Systeme sind einfacher zu verwalten

Entwickelt, um gemischte Work Loads zu verarbeiten

- Erlaubt Server Konsolidierung unter Beibehaltung eines Server pro Anwendung
- komplette Isolierung der verschiedenen Work Loads
- sehr schnelle Inter-Server Kommunikation

Scalability

- System z10 EC skaliert bis zu 64 Application Processors
 - bis zu 11 dedizierten I/O Processoren
- Hunderte bis tausende von virtuellen Linux Servers (abhängig von der Auslastung)

Wert von Linux auf System z

reduzierte Total Cost of Ownership (TCO)

- Environmental savings
 - ein Server-Rack statt mehrere hundert Server
- Consolidation savings
 - weniger Speichermedien, weniger Server, weniger Software licenses, weniger Server Management und Wartung

Verbesserte Servicefähigkeiten

- System Management (single point of control)
- Zuverlässigkeit, Verfügbarkeit, Sicherheit von System z
- Hoch performante Integration mit z/OS, z/VSE, z/TPF

Speed to Market

- Capacity-on-demand Fähigkeit von System z
- Dynamische Einrichtung eines neuen Servers es werden weniger als 10 Sekunden benötigt für ein neues Linux Server Image mit z/VM and IBM DS8000

Zusätzliche Anwendungen durch das Linux Application Portfolio

Große Anzahl von gutausgebildeten Programmierer, die mit Linux vertraut sind

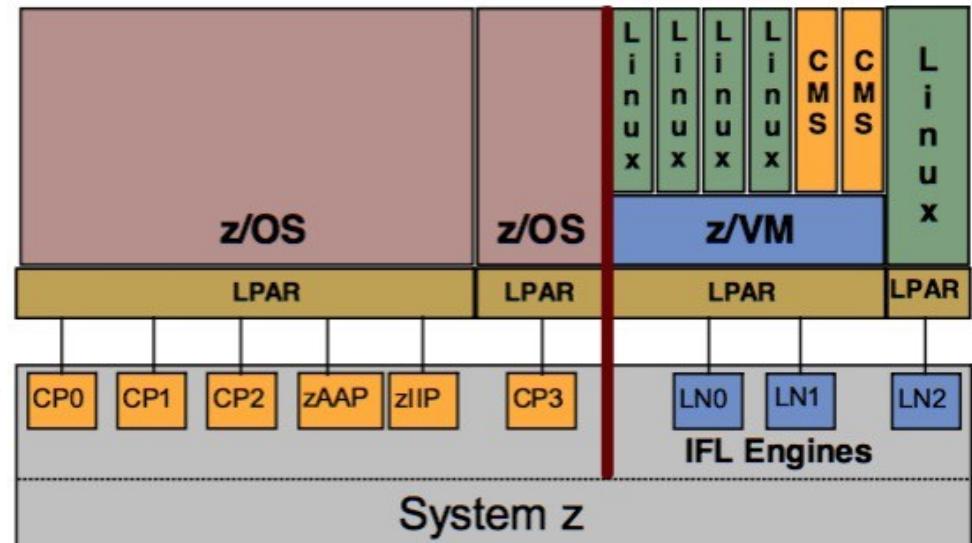
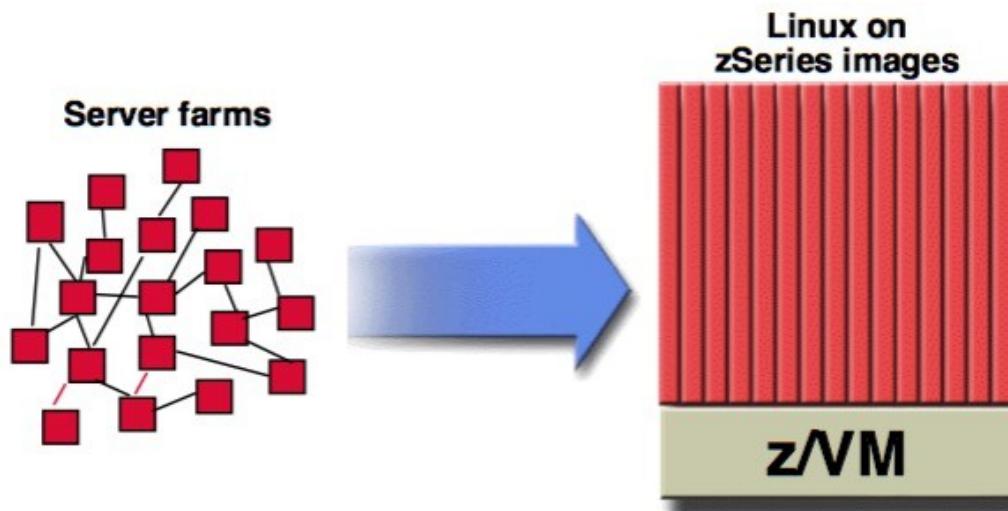
Integrierte Business Lösungen

- Daten-Vielfalt und -Verfügbarkeit auf System z
- weiter Bereich von Linux Anwendungen

System z – die ultimative Virtualizierungsplattform

Massive Konsolidierungsplattform

- 80+ logical partitions, 100-te bis 1000-de von virtuellen Servern mittels z/VM
- Virtualisierung ist integraler Teil des Systems, kein add-on
- HiperSockets für memory-speed Kommunikation
- Verfügt über die ausgefeiltesten und vollständigen Hypervisor Funktionen
- Intelligentes und autonomes Management von verschiedenen Workloads and System Resources basierend auf Geschäfts-policies und Workload Performance Erwartungen

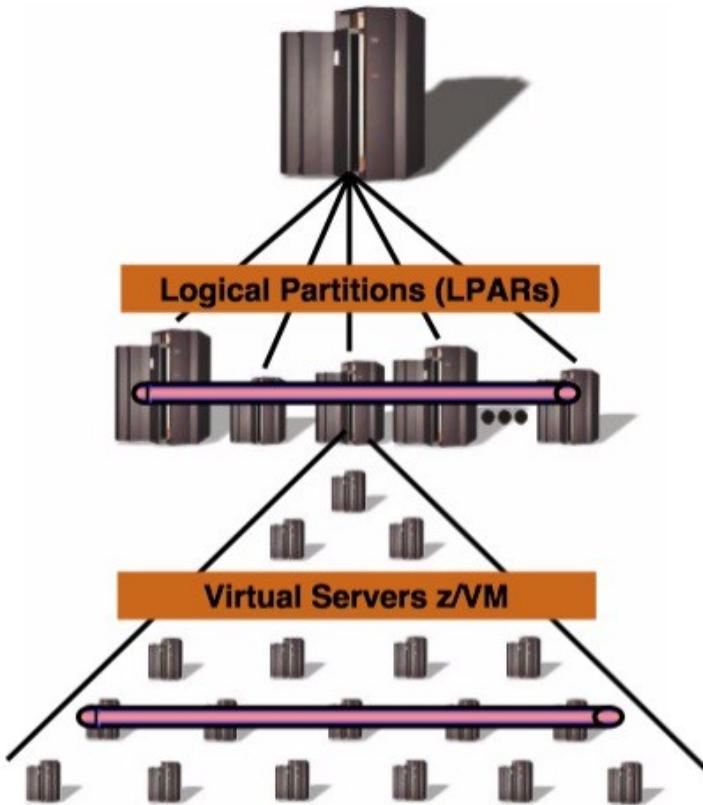


VM kann die Hardware über mehrere Ebenen virtualisieren und kann tausende von Images gleichzeitig ausführen:

- **TestPlan Charlie (2000):**
41,400 Linux images in an LPAR on a G5
- **TestPlan Omega (2002):**
97,943 Linux images on a Z7,
12-way@160 MIPS each, 16G RAM

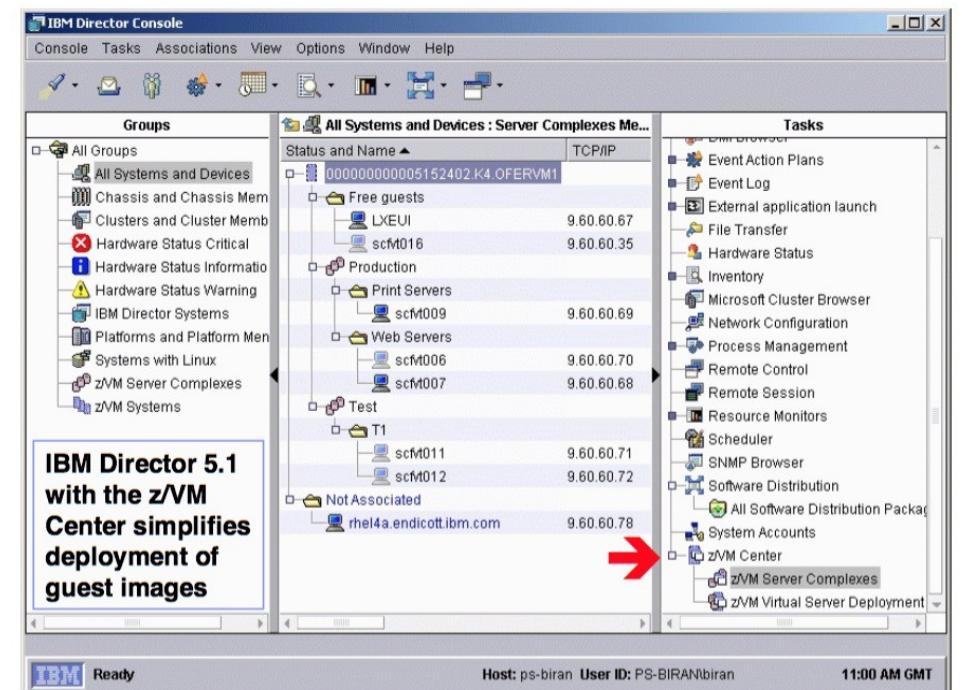
<https://administratosphere.wordpress.com/2009/11/02/test-plan-charlie-41000-linux-servers-on-one-box/>

z/VM – unbegrenzte Virtualisierung



z/VM 5.2 – 64-bit Unterstützung real und virtuell

- Ausgereifte Technologie – z/VM wurde 1967 eingeführt
- Software Hypervisor integriert in Hardware
 - Gemeinsames Teilen der CPUs, Speicher und I/O Einheiten
 - Virtuelles Netzwerk mit virtuellen Switches, Routers, etc.
 - Virtuelles I/O (mini-disks, virtual cache, ...)
 - Virtualisierte Middleware und Services (SNA/NCP, etc.)
- Einfaches Management
 - Schnelle Installation von neuen Servern
 - entweder durch **cloning** oder
 - mit der **z/VM Center** Funktion der IBM Director Konsole
 - Selbstoptimierendes Workload Management
 - Flexible Lösungen für Entwicklungs-, Test- und Produktionssysteme



Probleme mit Linux als Guest unter z/VM

[How to turn a Pinguin into a Dog](#)

Wenn man ein Betriebssystem, das ursprünglich entwickelt worden war, um auf einer realen Hardware (native) zu laufen, in eine virtuelle Umgebung packt, dann müssen Mechanismen und Regeln zum Teil anders definiert und angewendet werden, damit das Ergebnis optimal performant.

Jiffies

Der Begriff **Jiffy** bedeutet ein unbestimmtes, aber sehr kurzes Zeitintervall. In Betriebssystemen wie Unix und Linux ist ein Jiffy die Zeit zwischen 2 Timer-Ticks, der historisch bei 10ms lag. Diese Timer-Ticks geschehen unabhängig von der Workload des Servers und damit auch, wenn das System nichts zu tun hat, d.h. idle ist. Für den Hypervisor ist damit aber das System nicht idle, weil alle paar Millisekunden eine Reihe von Maschineninstruktionen ausgeführt werden. Aber der Linux Gast muss wirklich idle gehen, wenn nichts zu tun gibt, damit z/VM die Ressourcen des Gastes (z.B. die CPUs, den Arbeitsspeicher, etc.) freigeben kann und sie anderen Gästen zuteilen kann.

Jiffies werden richtig gesetzt durch folgende Anweisung: `echo 0 > /proc/sys/kernel/hz_timer`

32-bit Version ist besser

64-bit Guestsysteme benötigen per-se mehr Resources (e.g. Page Tabellen sind doppelt so groß) als ihr 32-bit Äquivalent, ohne das der Benutzer einen Vorteil hat, solange er nicht mehr als 2GB virtueller Speicher benötigt wird

Swapping ist gut

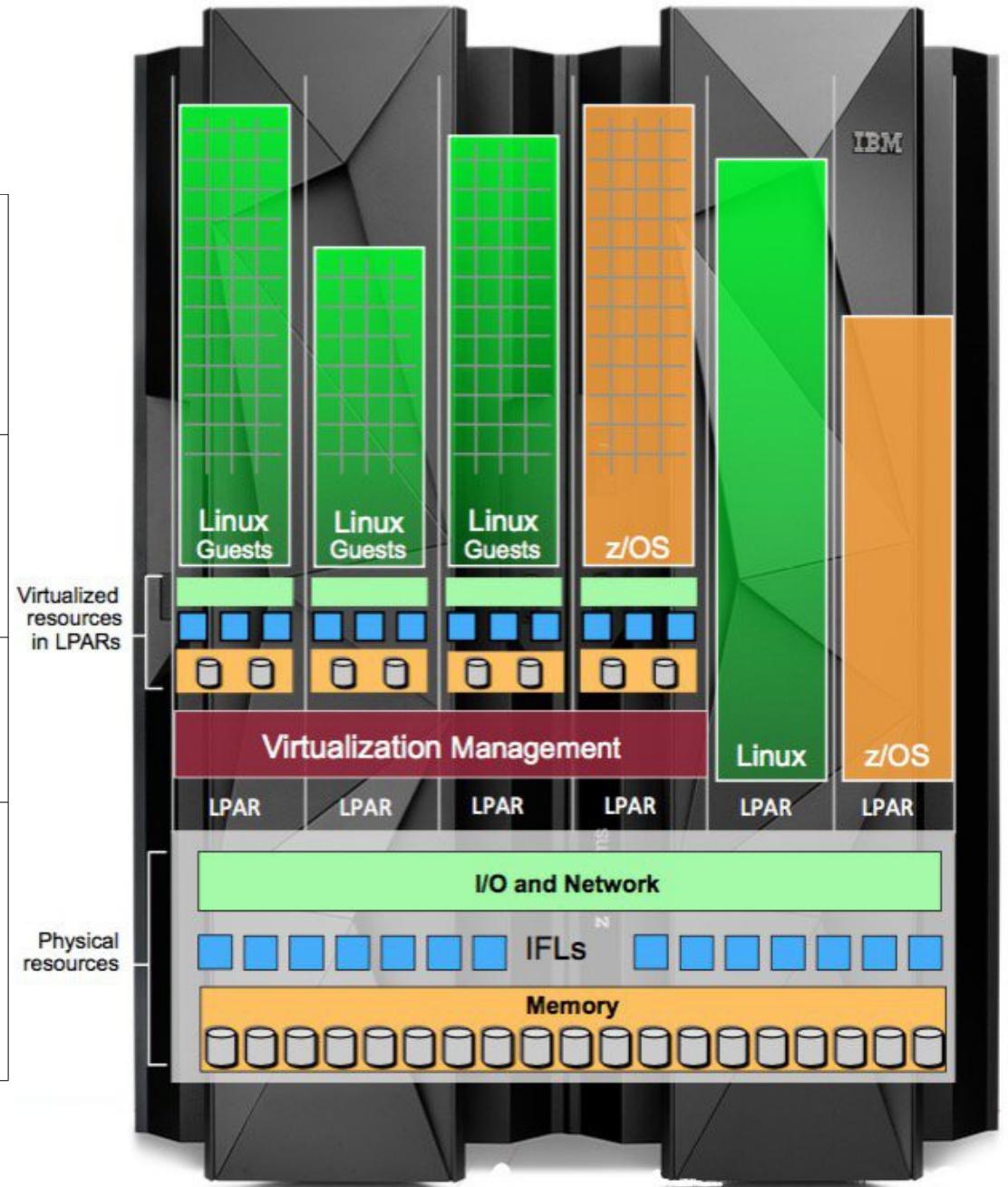
Normalerweise bedeutet Swapping, dass nicht benötigte Seiten auf das Swapping Device ausgelagert werden bzw. neue benötigte Seiten in den Arbeitsspeicher geholt werden. Beim Swapping finden eine Reihe von zeitintensiven I/O Operationen statt, die man nach Möglichkeit vermeiden will. Deshalb definiert man den benötigten Arbeitsspeicher so gross wie den maximalen Working Set der Anwendung, um Swapping zu vermeiden. Das führt aber zu unnötig großem Resourceverbrauch.

Bei zLinux unter z/VM kann das Swapping Device als virtuelles I/O Gerät (sogenannte VDISK) definiert werden und damit unterliegt es dem Paging Subsystem von z/VM und kann mit dem Paging-Verhalten der anderen Guestsysteme korreliert werden. Dadurch können die Server trotz Swapping besser performen als mit großen Arbeitsspeicher.

Prinzipiell ist es immer notwendig, das **Verhalten des Gesamtsystems bei verschiedenen Belastungen vor und nach Tuning-Maßnahmen zu messen**, um eine sinnvolle Aussage über den Effekt der Maßnahme machen zu können.

System z Platform Virtualisierung

LPAR	<p>Logical Partition Untermenge der physikalischen Resources virtualisiert als separate Computer</p> <p>bis zu 85 LPARs können konfiguriert werden</p>
IFL	<p>Integrated Facility for Linux Attribut für einen Core</p> <p>bis zu 141 Cores können als IFLs auf einer z13 definiert werden</p>
Virtualization Management	<p>Hypervisor mit Virtualization Management Funktionen für einfache Administration, Zurverfügungstellung (Provisioning) und Automatisierung</p>
Linux Guest	<p>virtuelle Linux Gastsysteme führen Workloads aus wie Analytics, WebHosting, Datenbanken, Java Apps, etc.</p> <p>bis zu tausend Linux Gäste können auf einer einzelnen z13 existieren</p>



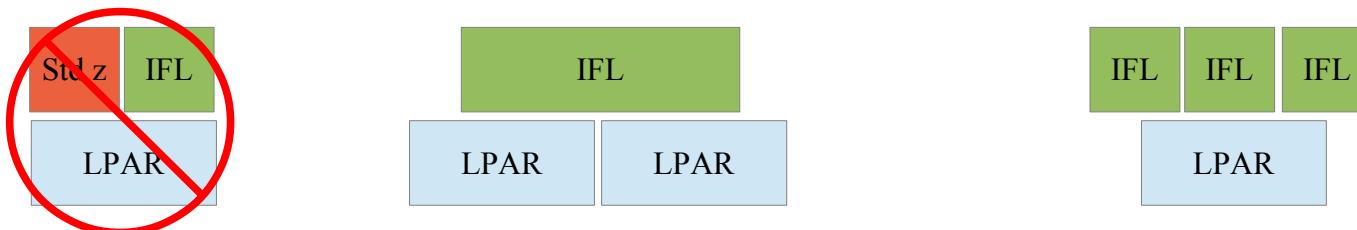
Integrated Facility for Linux (IFL)

Es gibt viele Möglichkeiten, Linux Anwendungen auf einem Mainframe auszuführen:

- als Job unter z/OS unter Verwendung der Posix1003.2-compliant Unix System Services USS
- Linux als einiges Betriebssystem in einer System z logischen Partition (LPAR)
- Linux als Gast-Betriebssystem unter z/VM auf System z CPUs
- Linux als Gast-Betriebssystem unter z/VM auf speziellen Linux Prozessoren

Die **Integrated Facility for Linux (IFL)** wurde im September 2000 eingeführt, um gezielt die Softwarekosten für Linux Anwendungen zu reduzieren, ohne auf die Einnahmen von den traditionellen Anwendungen auf z/OS zu verzichten. Dafür wird ein System z Prozessor so aufgesetzt, das er das Linux Betriebssystem mit oder ohne z/VM ausführen kann. Andere System z Betriebssysteme wie z/OS, z/VSE, etc. können von diesem Prozessor nicht ausgeführt werden. Ein IFL ist nicht notwendig, um zLinux auszuführen, aber Kosten für die Ausführung von zLinux Anwendungen in einer IFL sind signifikant niedriger als auf einem normalen z Prozessor, die von der Leistungsfähigkeit (MIPS – Million Instruktionen Per Second) bestimmt wird. Ein IFL Prozessor kostet eine einmalige Gebühr con ca. 40K\$.

Eine IFL ist nur im LPAR Mode benutzbar und kann nicht mit standard System z Prozessoren gemischt werden:



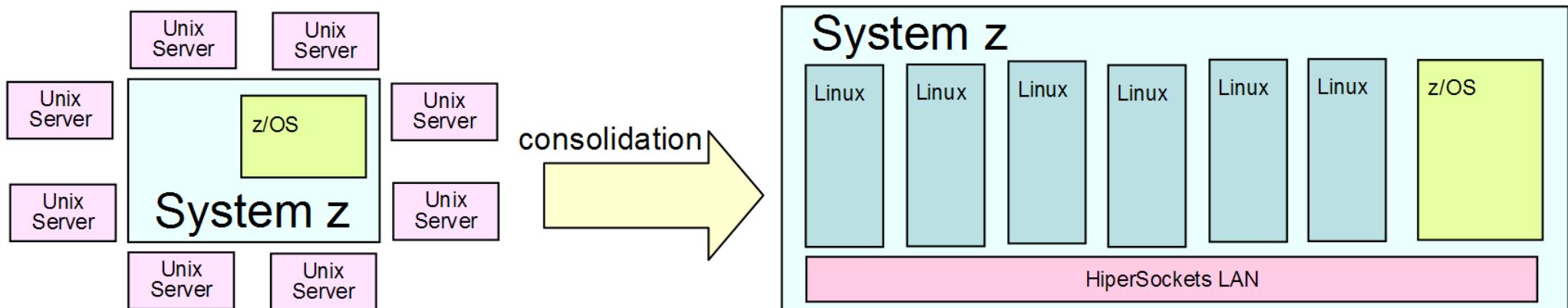
Server Konsolidierung mit zLinux

In vielen Unternehmen und staatlichen Organisationen existieren neben dem Mainframe hunderte oder tausende von dezentralisierten Servern. Diese Server sind vielfach über ein historisch gewachsenes Netzwerk von Ethernet Verbindungen, Switches und Routern miteinander verbunden. Um die Administrationskosten zu senken ist es wünschenswert, diese Agglomeration von Servern zu zentralisieren.

Angenommen, auf allen dezentralisierten Servern läuft Linux als Betriebssystem. Es ist möglich, die dezentralen Server abzubauen und stattdessen eine Reihe von „Linux on System z“-Instanzen auf einem größeren Mainframe Server einzurichten.

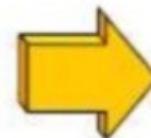
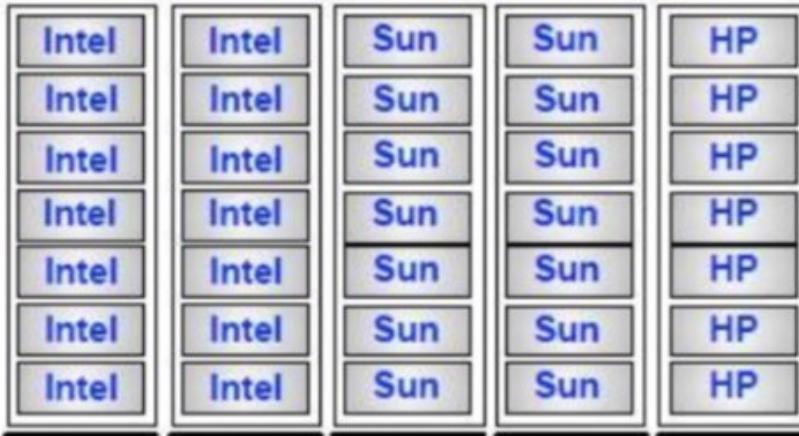
Die CPU Auslastung auf dezentralen Servern liegt häufig bei unter 20 %. Da „Linux on System z“-Prozessoren mit einer höheren Auslastung (bis zu 100 %) betrieben werden können, kann die Anzahl der „Linux on System z“-Instanzen deutlich kleiner als die Anzahl der bisherigen Linux Server sein.

Die Firma IBM macht Reklame mit dem relativ niedrigen Energie Verbrauch und der Umweltfreundlichkeit ihrer Mainframe Server. In Bezug auf Linux Server Konsolidierung ist dies berechtigt.

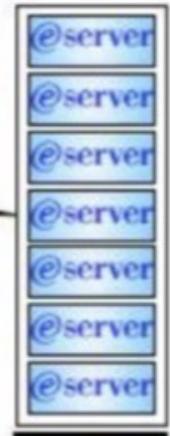




Traditional Server Farm



Server Farm in a Box



- separate Server verursachen zusätzliche Kosten
 - Hardware Kosten und Wartung
 - zusätzliches Bedienungspersonal
 - Raumfläche, Strom, Kühlung
 - Software Lizenzen pro Server
- Kommunikationsverbindungen erfordern Kilometer von Kabel
- Hohe Verfügbarkeit wird durch zusätzliche System bzw. durch Re-Boots sichergestellt

- ein Server pro Anwendungen reduziert Kosten ohne die Serverautonomie zu verletzen
- Hochgeschwindigkeitskommunikation durch virtuelles LAN
- bekommt Hochverfügbarkeit durch die Architektur-inherenten Funktionen und Dienste
- Mainframe Qualitäten bezüglich Zuverlässigkeit und Wartung für alle Server

Rezentralisierung mit „Linux on System z“

Mit der Rezentralisierung zahlreicher Linux Server auf dem Mainframe kann die komplexe Ethernet Infrastruktur vereinfacht werden. Da physische Ethernetkabel durch virtuelle HiperSocket-Verbindungen ersetzt werden, ist ein Abhören der Verbindungen (Man-of-the-Middle-Attacke) nicht mehr möglich. Komplexe Verschlüsselungsverfahren können entfallen. Separate Firewall Rechner und mehrfache Demilitarized Zones (DMZ) sind möglicherweise ebenfalls nicht mehr erforderlich. Die physische Ethernet Struktur wird durch virtuelle LANs ersetzt und dabei vermutlich wesentlich vereinfacht.

Über HiperSockets können die „Linux on System z“-Instanzen Dienste des z/OS Security Servers wie RACF und LDAP nutzen. Es ist möglich „Linux on System z“-LPARS auf unterschiedlichen Systemen eines Sysplex über HiperSockets und XCF miteinander zu verbinden.

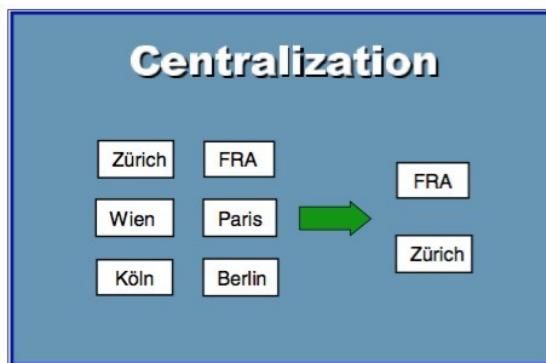
Mit zLinux ist es möglich, eine große Anzahl von physikalischen Servern, die jeweils mit geringer Auslastung laufen, auf eine gemeinsame Hardware zu konsolidieren.

Die Gründe für die Konsolidierung liegen in der Senkung der allgemeinen Betriebskosten (**Total Cost of Ownership TCO**):

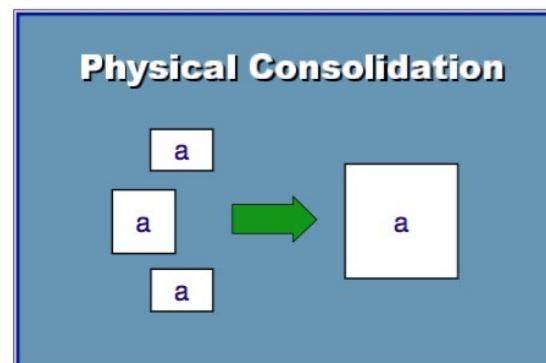
- Steigerung der Auslastung der Server Hardware, Software und der Netzwerkinfrastruktur
- Senkung der Kosten für Personal, Räume und Energie
- Steigerung der Verfügbarkeit und damit geringere Kosten für Ausfallzeiten
- schnelleres Zurverfügungstellen von neuen Servern.

Es gibt unterschiedliche Typen von Konsolidierung:

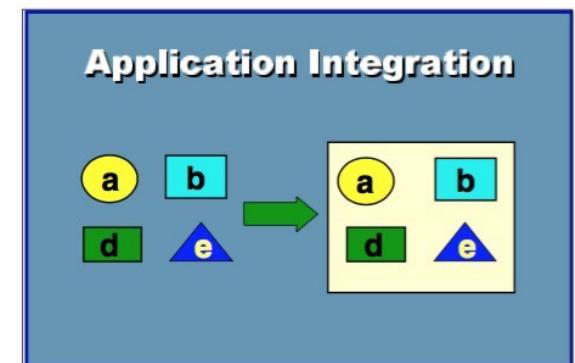
Zentralisierung



Physikalische Konsolidierung



Integration der Anwendungen



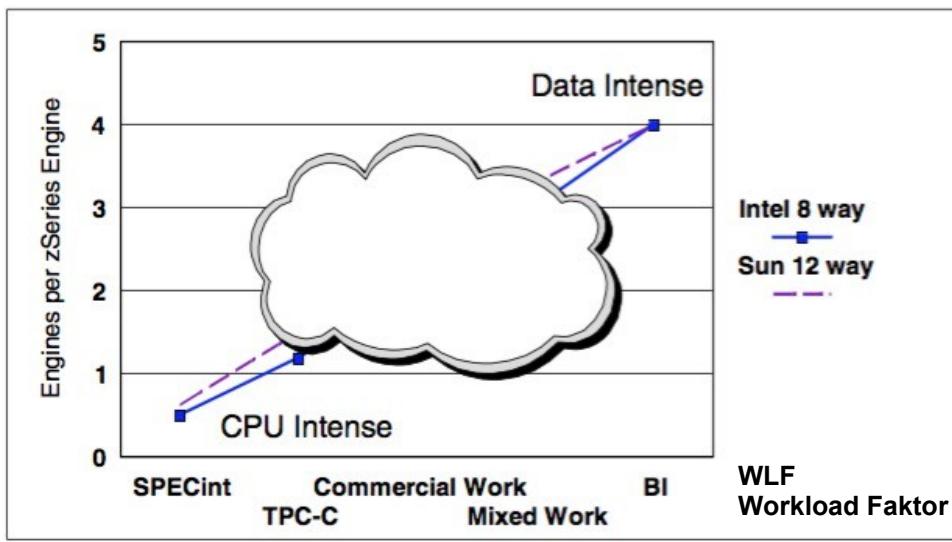
Außerdem unterscheidet man zwischen

- **vollständiger Konsolidierung ($n \rightarrow 1$)**: mehrere Server, die entweder alle die gleiche oder verschiedene Anwendungen ausführen, werden auf einen Server mit einem Betriebssystem Image zusammengefaßt.
- **virtuelle Konsolidierung ($n \rightarrow n$)**: mehrere physikalische Server werden auf einer Hardware in die gleiche Anzahl von Guest Machines auf einer Virtualisierungsplattform installiert.

Wenn zur Server Konsolidierung zLinux verwendet wird, dann sind folgende Punkte zu beachten:

- **ISV Software**: nicht alle Software von Drittanbietern ist auf zLinux verfügbar
- **Netzwerk Bandbreite**: insbesondere bei Zentralisierung ist mit einer erhöhten Netzwerkauslastung zu rechnen
- **Personaleinsparungen**: solange die Organisation und die Geschäftsprozesse nicht mitangepasst werden, ist mit eher geringen bis gar keinen Einsparungen zu rechnen.

Bei der Abbildung einer Linux/Intel Server Farm nach Linux auf einer System z Mainframe ist zu berücksichtigen, wieviele Intel Boxen einer System z CPU entsprechen. Leider gibt es dafür keine einheitliche Maßzahl, sondern es hängt von der Art der Workload des zu konsolidierenden Systems ab:



Diese Graphik zeigt, dass im besten Fall (rechts oben) eine zCPU die Workload von etwa 2-3 SUN/Intel CPUs bei 100% Auslastung leisten kann. Geht man davon aus, dass die Auslastung von verteilten Servern in der Regel eher bei 10% liegt, so bedeutet dies, dass eine zCPU die Workload von etwa 30-40 Sun/Intel CPUs bearbeiten kann.

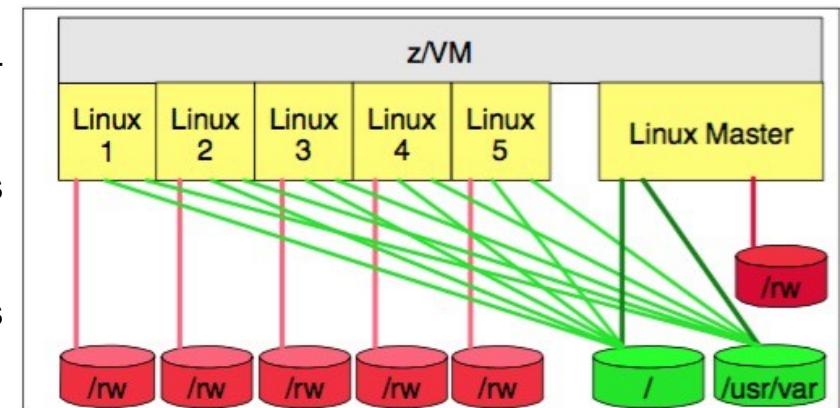
Als Faustformel zur Berechnung der notwendigen Leistung des Mainframe gilt:

$$\text{Processors}_z = (\text{Utilization}_{\text{other}} * \text{Processors}_{\text{other}} * \text{MHz}_{\text{other}}) / (\text{MHz}_z * \text{WLF})$$

Das bedeutet, dass die Anzahl der notwendigen zCPUs von der Art der Workload und den Charakteristiken und der Auslastung der Ursprungssystemen abhängt.

Server Farming mit z/VM und zLinux

Für die virtuelle Konsolidierung auf System z wird z/VM als Virtualisierungs-Plattform verwendet und dann für jeden physikalischen Server eine zLinux Instanz installiert. Die zLinux Systeme sind so aufgesetzt, dass sie alle Klone eines Masters sind. Diesem Master gehört ein R/O Filesystem, das ca. 90% der Files beinhaltet, die zum Betrieb eines Linux Systems notwendig sind. Über symbolische Links greifen die zLinux Klone auf die R/O Files zu. Für die verbleibenden 10% hat jedes Linux System ein eigenes R/W Filesystem, dass mit Hilfe von Installationsskripten gefüllt wird.



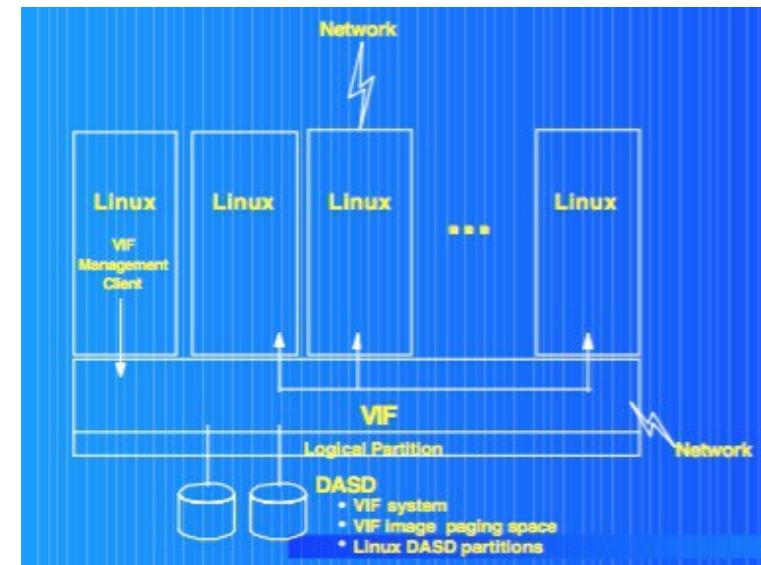
Den Effekt der Server Konsolidierung zeigt folgendes Video: [IBM -The Heist](#)

Linux-only auf System z

Für Neukunden, d.h. für Kunden, die noch kein System z in ihrer IT Installation haben, ist zLinux auf Grund der Verfügbarkeit und der günstigen TCO sehr attraktiv. Als Nachteil galt aber, dass zur Installation und zum Betrieb von zLinux auf System z das Wissen über Mainframe Software notwendig war. Dieses Problem wurde zuerst mit der Virtual Image Facility und heutzutage mit KVM adressiert.

Die **Virtual Image Facility (VIF) for Linux** war eine stripped-down Version von VM, welches es ermöglichte, mehrere Linux System auf einem einzigen S/390 System auszuführen.

- VIF lief entweder direkt (native) auf der S/390 Hardware oder in einer LAPR Partition
- es benötigte zum Betrieb keine VM Kenntnisse, obwohl es ein Derivat von zVM war
- unterstützte mehrere Hunderte von Linux Images, abhängig von der Workload
- geringere System Management Fähigkeiten als z/VM mit geringerer Granularität
- sehr attraktive Preiskonditionen basierend auf der Anzahl der installierten Linux Systemen
- konnte nur S/390 Linux (i.e. 32-bit Linux) ausführen



Da die Funktionalität und Flexibilität von VIF gegenüber dem Standardprodukt z/VM eingeschränkt war und es trotzdem notwendig war, spezielles Wissen zu besitzen, um ein Linux-only Sysstem mit Hilfe von VIF zu betreiben, wurde das Produkt nach ein paar Jahren wieder vom Markt genommen.

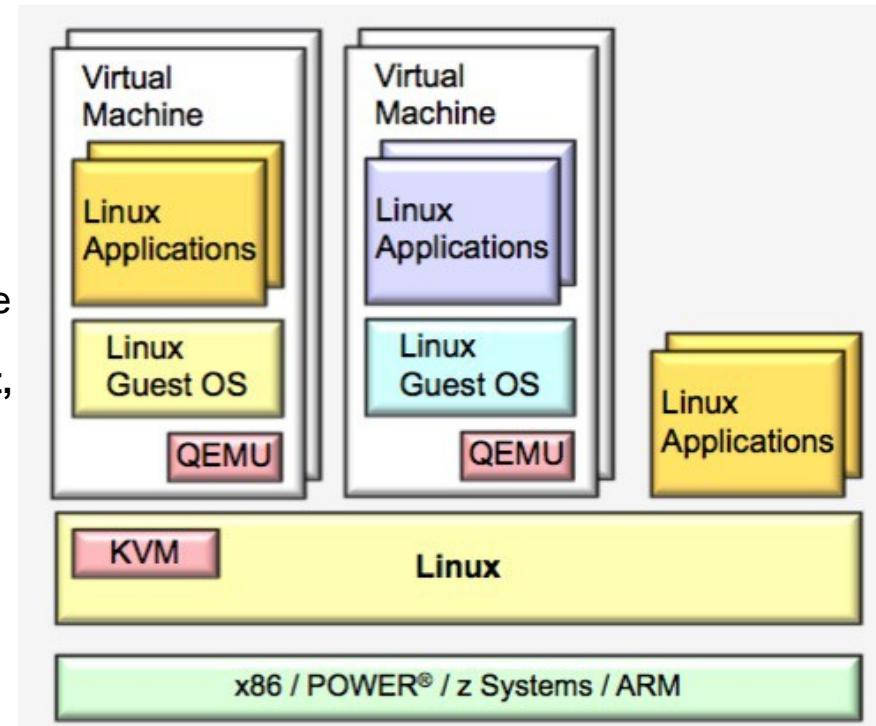
KVM (Kernel-based Virtual Machine)

Um die Abhängigkeit von System z Software zu eliminieren, wurde ab 2010 ein Entwicklungsprojekt im Labor Böblingen gestartet, um eine reine Linux Virtualisierung zu erreichen, die von den Kunden kein z/OS Know-How verlangt.

Zu dem Zwecke wurde daran gearbeitet, das **Linux Kernel Modul KVM** auf System z Hardware zu portieren.

KVM hat folgende Eigenschaften und Komponenten:

- **KVM stellt eine Schnittstelle zur Verfügung, um virtuelle Maschinen zu installieren und zu betreiben**
 - Es ist ein Linux Kernel Module, welches einem User Space Programm erlaubt, die Hardware Virtualization Features von verschiedenen Processoren zu benutzen
 - Damit wird ein regulärer Linux Kernel ein Virtual Machine Monitor (VMM, Hypervisor)
 - KVM selbst ist keine Emulation einer physikalischen Hardware
- **QEMU (Quick Emulator) ist ein weiteres OpenSource Projekt, das benutzt werden kann, um die physikalischen Hardware dem Gastsystem zur Verfügung zu stellen**
 - QEMU unterstützt KVM, wenn es dieselbe Zielarchitektur wie die Host Architektur verwaltet
 - ein KVM/QEMU Benutzerprozess mit einem Linux Guest ist einfach ein weiterer Benutzerprozess für Linux
 - unterstützt z/Architecture, Power, x86, ARM, MIPS32, usw.
- **Auf System z läuft mit KVM/QEMU ein virtuelles Linux System, welches Virtual I/O benutzt**
 - Bis jetzt gibt es keine vollständige Unterstützung für System z I/O wie z.B. Start Subchannel
 - KVM läuft nur in einer LPAR Partition, nicht als z/VM Guest benutzt die **Start Interpretive Execution SIE** Instruktion, die auch von LPAR benutzt wird (One Level Nesting only)



KVM/QEMU für System z erlaubt es nur, zLinux Gastsysteme unter zLinux auszuführen!

KVM Infrastruktur / Hypervisor Management

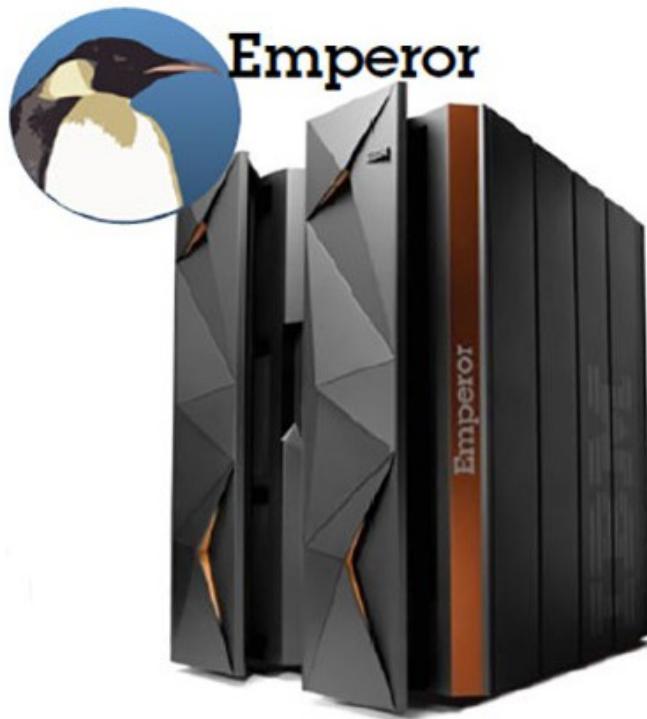
Die virtuellen zLinux Gastmaschinen werden mit Hilfe von **virsh** Command Line Interface (CLI) Kommandos verwaltet.
Zu den Aufgaben, die ein Linux HostOS / Hypervisor Administrator ausführt, um das System zu managen, gehören:

- Booten / Shutdown des Host Betriebssystem
- Verwalten der System Resources
 - laden und entladen von Kernel Modules
 - konfigurieren des Startup Demons systemd
 - Hotplug Memory und CPUs
 - automatisierte System Tasks
- Installieren der Security und Crypto Einrichtungen
 - Firewalls, SELinux (Security Enhanced Linux)
- Verwalten von Benutzern und Benutzergruppen
- Konfiguriere das Speichersystem
 - formatiere und partitioniere die Platten
 - konfiguriere den Weg zu den Geräten inklusive den Multipath Zugang
 - Verwaltung der Filesystem mit Hilfe des Logical Volume Managers LVM
- Konfiguriere das Netzwerk
 - angeschlossene Geräte wie Router, Switches, etc.
 - administriere die Verbindungen zwischen den Gästen untereinander und zum Host
- Aufsetzen der First Failure Data Capture (FFDC)
Problem Determination
 - definiere das Systemverhalten bei System Panic
 - sammle und zeige sosreports, Logs, Dumps, etc.
- Konfigurieren des yum Repository für rpms, um System Updates zu erhalten
- Leistungsmessungen und Diagnose
- Verwalten von Client-seitigen Dienste wie DNS, DHCP, OpenLDAP, etc.

Alle die Funktionen und Services können mit Hilfe von Linux eigenen Programmen und Skripten ausgeführt werden, die jeder gute Linux Systemadministrator kennt und beherrscht. Es ist kein zusätzlicher System z Softwareskill notwendig!

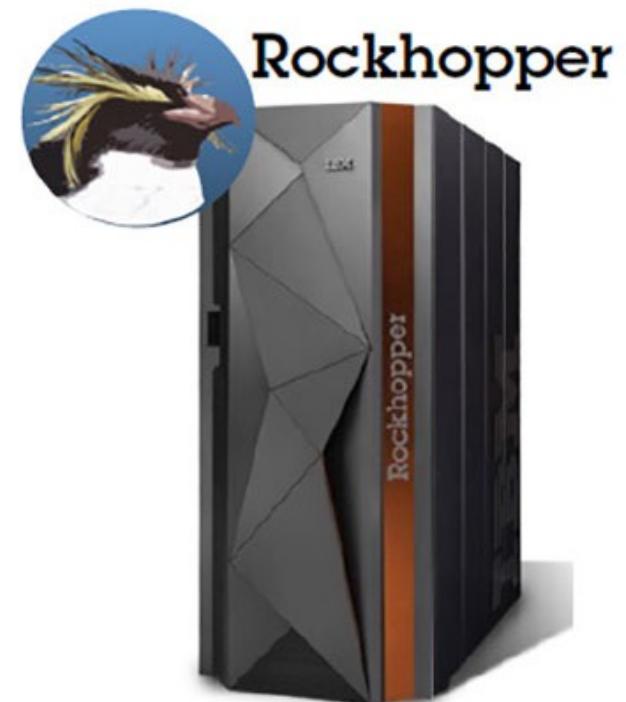
LinuxONE

Basierend auf KVM / QEMU und zLinux hat IBM 2 neue **Linux-only auf System z** Produkte 2015 angekündigt:



Emperor

Während der **Kaiserpinguin (Emperor)** ein HighEnd Modell der neuesten IBM System z13 System ist, basierte das kleinere System **Felsenpinguin (Rockhopper)** ursprünglich auf der zEC12. Inzwischen ist es auch auf die kleinere Version der z13 portiert worden.



Rockhopper

Der Unterschied liegt in der Ausstattung und damit in der Leistungsfähigkeit der Systeme:

	Emperor	Rockhopper
Anzahl von CPU Cores	141	20
Frequenz	5 GHz	4,3 GHz
maximaler Hauptspeicher	10 TB	4 TB

Unix und Linux auf System z Teil 5

Weiterführende Informationen

Literatur zum Thema „Multics“

<http://www.multicians.org/papers.html>

[Introduction and Overview of the Multics System](#)

[Honeywell Multics System 1975](#)

Literatur zum Thema „Unix“

der Klassiker unter den Unix Artikel von den Entwicklern Dennis M. Ritchie und Ken Thompson:

[The UNIX Time-Sharing System](#)

Literatur zum Thema „Unix System Services“:

[ABCs of z/OS System Programming Volume 9, November 2005, IBM Form No. SG24-6989-01](#)

[z/OS UNIX System Services User 's Guide, IBM Form No. SA22-7801-04](#)

Literatur zum Thema „Linux“

[Wikipedia Entry for Linux](#)

[History of Linux](#)

Linux Foundation <https://www.linux.com/>
<http://www.linuxfoundation.org/>

Literatur zum Thema „zLinux“

[Linux for s/390 vom Marist College, NY](#)

[Presentations on Linux/390](#)

Distributoren:

Red Hat [Red Hat Linux for zSeries](#)

SuSE [SuSE Linux Enterprise Server for IBM zSeries](#)

[Redbook Paper - Server Consolidation with Linux for zSeries](#)