www.uni-frankfurt.de

Modul: Programmierung B-PRG
Grundlagen der Programmierung 1

V23 Services des Betriebssystems

Prof. Dr. Detlef Krömker
Professur für Graphische Datenverarbeitung
Institut für Informatik
Fachbereich Informatik und Mathematik (12)

Unsere heutigen Lernziele

Übersicht erhalten über Betriebssysteme, d.h. Beantwortung der folgenden Fragen:

- Wozu ist ein Betriebssystem da?
- Welche Aufgaben muss es erfüllen?
- Mit welchen Konzepten tut es dies?
- Und zwar möglichst unabhängig vo konkreten Betriebssystemen

Etwas Historie zu Betriebssystemen erfahren

Wie kann ich diese Betriebssystem-Funktionen aus Python heraus nutzen.

Übersicht

- Aufgabe von Betriebssystemen
- Historische Entwicklung von Betriebssystemen
- Unterschiedliche Arten von Betriebssystemen
- Das Betriebssystem aus Sicht von Python

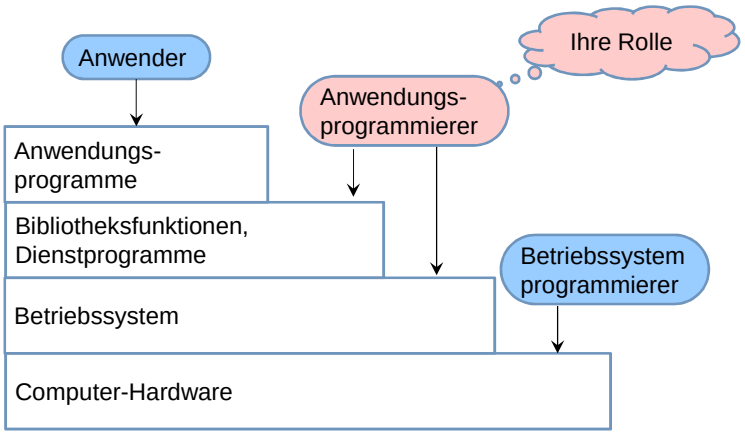
Aufgaben von Betriebssystemen

- **Betriebssystem = Computerprogramm**
- mit zwei grundsätzlichen Funktionen
  - **Abstraktion von der tatsächlich vorhandenen Hardware, d.h.**
    1. Abstraktion von der CPU / den CPUs **durch** Prozesse und Threads
    2. Abstraktion vom Arbeitsspeicher **durch** Speicherverwaltung
    3. Abstraktion von der Platte **durch** Dateisysteme
    4. Abstraktion von anderen Ein-/Ausgabesystemen
    5. Abstraktion vom Netzwerk
  - **Verwaltung von Systemressourcen**

Bereitstellen einer erweiterten Maschine

- Ziel: Abstraktion von detailliertem Verhalten des zugrunde liegenden Rechners auf möglichst hohem Niveau.
- Verstecken realer Hardware-Eigenschaften vor dem Benutzer.
- Das Betriebssystem stellt der Programmierer*in eine abstrakte Programmierschnittstelle zur Verfügung.
- Das Betriebssystem bietet einen Satz von Kommandos (Systemaufrufen), über die z.B. auf Ein-/Ausgabegeräte zugegriffen werden kann.
- Soll komfortabel für die Programmierer*in sein.

Schichtenarchitektur



Dienstprogramme

- Schnittstelle zwischen Betriebssystem und Dienstprogrammen nicht
- immer klar definiert

- Typische Dienstprogramme:
  - Compiler
  - Editoren
  - Kommandointerpreter (sog. Shell)

Verwaltung von Systemressourcen

- Ziel: Verwaltung aller Bestandteile eines komplexen Systems (Betriebsmittel)
- Beispiele: Prozessoren, Speicher, Platten, Netzwerkschnittstellen, Drucker, etc..
- Betriebssystemaufgabe: Geordnete und gerechte Zuteilung der Betriebsmittel an konkurrierende Prozesse / Benutzer.
- Auflösung von Konflikten bei der Betriebsmittelanforderung
- Schutz verschiedener Benutzer gegeneinander (z.B. Zugriffskontrolle bei Dateien)
- Effiziente Verwaltung von Betriebsmitteln

Betriebssystem als Ressourcenmanager (2)

- Fehlererkennung, Fehlerbehandlung
  - Hardware: Gerätefehler
  - Software: Programmfehler
- Ressourcenverwaltung in zwei Dimensionen:
  - Zeit: Verschiedene Benutzer erhalten Betriebsmittel nacheinander
  - Raum: Verschiedene Benutzer erhalten verschiedene Teile einer Ressource (z.B. Hauptspeicher)

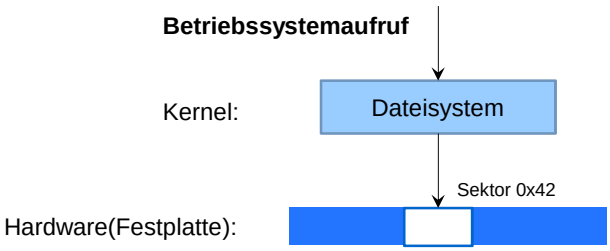
Der BS (OS) Kernel

- Der Kernel (Kern) des Betriebssystems wird in einem privilegierten Modus ausgeführt.
- Zugriff auf Ein- und Ausgabegeräte
- Speicherschutz veränderbar
- Benutzerprogramme müssen Systemaufrufe (*system calls*) ausführen, um diese Dienste zu nutzen.
- Nach Abarbeiten des Systemaufrufs übergibt das Betriebssystem die Kontrolle zurück an das Benutzerprogramm.

Betriebssystemaufruf aus einem Anwendungsprogramm

- Anfrage des Benutzers nach Inhalt einer Datei muss vom Betriebssystem beantwortet werden

Öffne Datei `/home/someuser/Documents/systeme1.txt`



Zugriff auf Ein- und Ausgabegeräte

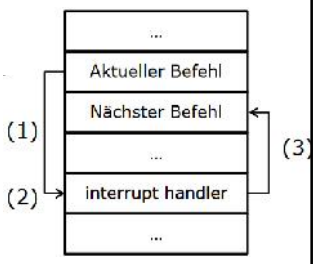
Es existieren verschiedene Methoden, auf Ein- und Ausgabegeräte zuzugreifen:

1. „synchroner“ Aufruf: der Systemaufruf fragt in einer Endlosschleife ab, ob die Operation beendet ist: **aktives Warten**
2. Gerät ruft eine Interrupt-Routine auf, so bald es fertig ist
3. Direct Memory Access über speziellen DMA-Chip

Betriebssystemaufruf durch Interrupts

- Geräte-Controller signalisiert dem Interrupt-Controller eine abgeschlossene Operation
- wird der Interrupt behandelt, so wird

1. Die aktuelle Programmausführung unterbrochen.
2. Eine Unterbrechungsroutine (interrupt handler) ausgeführt.
3. Die ursprüngliche Programmausführung fortgesetzt.



Erweiterbarkeit, Entwicklungsfähigkeit von Betriebssystemen

- Änderungen des Betriebssystems erforderlich durch z.B.
  - Neue Hardware
  - Neue Protokolle
  - Korrekturen (z.B. Schließen von Sicherheitslöchern)
- Eigenschaften eines Betriebssystems
  - Modular und klar strukturiert aufgebaut
  - Gut dokumentiert

Historische Entwicklung von Betriebssystemen (1)

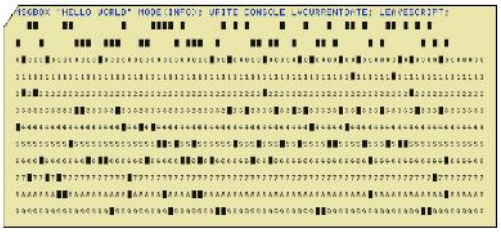
Verschiedene Entwicklungsstadien:

- Serielle Systeme
- Einfache Stapelverarbeitungssysteme
- Mehrprogrammfähige Stapelverarbeitungssysteme
- Timesharing-Systeme
- Systeme mit graphischen Benutzeroberflächen
- Netzwerkbetriebssysteme
- Verteilte Betriebssysteme

Historische Entwicklung von Betriebssystemen (1)

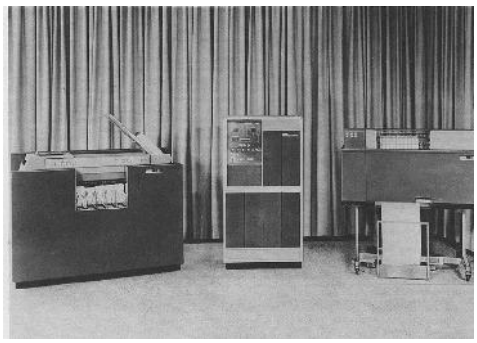
Serielle Systeme (1945-1955)

Single-User, Single Task Betrieb von Rechnern **ohne Betriebssystem**
"Programmeingabe" in Rechner über Lochkarten / Lochstreifen
Zuteilung von Rechenzeit durch Reservierung mit Hilfe Papieraushang



Historische Entwicklung von Betriebssystemen (2)

Einfache Stapelverarbeitungssysteme (ab 1955)

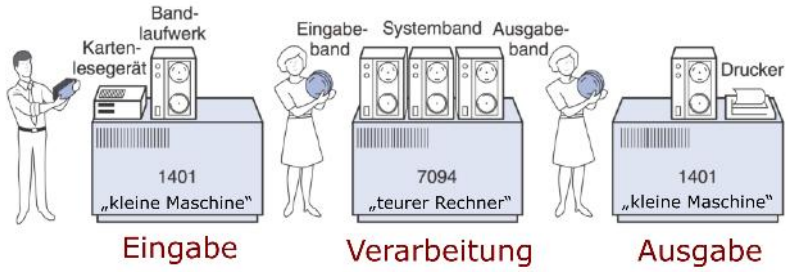


Fotograf: Gobierno de los Estados Unidos
Basic IBM 1401 system from BRL 1961.
"A Third Survey of Domestic Electronic Digital Computing Systems", Report No. 1115, March 1961 by Martin H. Weik, published by Ballistic Research Laboratories, Aberdeen Proving Ground, Maryland.
http://ed-thelen.org/comp-hist/BRL61-ibm1401.html#IBM-1401

Historische Entwicklung
Einfache Stapelverarbeitungssysteme (ab ca. 1955)

- Unterscheidung von Programmentwicklern und Operateuren, welche die Rechner („Mainframes“) betrieben und bedienten.
- Programmentwicklung auf Papier (in FORTRAN, Cobol, Assembler), Stanzen auf Lochkarten (Lochen).
- Sammlung von Programmen (Jobs) auf **Lochkartenstapeln.**
- Einlesen und Ausgeben der gesammelten Jobs mit "kleinen" Rechnern Speichern auf Band (nicht auf Platten!).

Historische Entwicklung (1955 – 1965)



Nach Wolfram Burgard (Uni fFeiburg): Kapitel 2
Überblick Betriebssysteme aus Systeme I: Betriebssysteme

Mehrprogrammfähige Stapelverarbeitungssysteme (ab 1965)

- Verarbeitung durch einen(!) Rechner (IBM 360 sowie Nachfolger)
- Spooling
  - Einlesen von Jobs auf Lochkarten, danach Speichern auf Platte
  - Nach Beenden eines Jobs: Laden eines neuen Jobs von Platte
- Mehrprogrammfähigkeit bzw. Multiprogrammierung, um Wartezeiten bei E/A zu reduzieren

Historische Entwicklung von Betriebssystemen (4) Timesharing-Systeme (ab Mitte 60er)

- Bis dahin Nachteil: Kein interaktives Arbeiten mehrerer Benutzer möglich
- Timesharing-Systeme: **Online-Zugang** zum System für alle Benutzer
- Idee: Interaktives Arbeiten eines Benutzers erfordert nicht die komplette Rechenzeit eines Rechners
- Bei schnellem Umschalten bemerkt der Einzelnutzer nicht, dass er die Maschine nicht für sich allein hat

Stapelverarbeitung	Timesharing
Maximale Prozessornutzung (Betreiberwunsch)	Minimale Antwortzeit (Benutzerwunsch)
Befehle in Jobsteuersprache	Interaktive Kommandos

Historische Entwicklung von Betriebssystemen (5) Systeme mit grafischen Benutzeroberflächen ab 80

- GUI (Graphical User Interface): Fenster, Icons, Menüs, Mauszeiger
- Zuerst übernommen durch Apple Macintosh
- Später durch Microsoft Windows 1985-1995: z.T. Grafische Umgebung, aufsetzend auf MS-DOS (Graphisch ab Windows 3.1)
- ab Windows 95: Betriebssystem und GUI stark miteinander verschränkt

Netzwerkbetriebssysteme (6) (ab 80er)

- Benutzer kennt mehrere vernetzte Rechner (Workstations)
- Einloggen auf entfernten Rechnern möglich
- Datenaustausch möglich
- Auf Einzelrechnern (Workstations, Personal Computer, PC): Lokales Betriebssystem, lokale Benutzer
- Netzwerkbetriebssystem = „normales Betriebssystem mit zusätzlichen Fähigkeiten“

Beispiel: Unix / LINUX:

- GUI "nur" als Aufsatz auf das Betriebssystem
- X-Window-System: Grundlegende Funktionen zur Fensterverwaltung
- Komplette GUI-Umgebungen basierend auf X-Window: z.B. KDE, GNOME

Historische Entwicklung von Betriebssystemen (7) Verteilte Betriebssysteme

- Mehrere vernetzte Rechner erscheinen dem Benutzer wie Einprozessorsystem
- Datenspeicherung und Programmausführung verteilt auf mehreren Rechnern
- Verwaltung automatisch und effizient durch Betriebssystem
- Probleme: Nachrichtenverzögerungen, Dateninkonsistenz

Historische Entwicklung von Betriebssystemen (8) Aktuell

- **Betriebssysteme für Mehrkern-Prozessoren**
  - Aufteilung der Prozesse auf vorhandene Kerne
  - Eigene Recheneinheiten, Zugriff auf gemeinsame Ressourcen
  - Theoretisch n-fache Rechenleistung bei n Kernen (abhängig von der Parallelisierung der Software)
- **Betriebssystem für mobile Geräte mit Touch-Bedienung** (Smartphones und Tablet-PC)
  - zahlreiche Minianwendungen, so genannte **Apps**.
  - starke Anleihen bei z.B. Unix (Android)

Übersicht

- Aufgabe von Betriebssystemen
- Historische Entwicklung von Betriebssystemen
- Unterschiedliche Arten von Betriebssystemen
- Das Betriebssystem aus Sicht von Python

Arten von Betriebssystemen (1)
Mainframe-Betriebssysteme

- Betriebssysteme für Großrechner
- Einsatz: Webserver, E-Commerce, Business-to-Business
- Viele Prozesse gleichzeitig mit hohem Bedarf an schneller Ein-/Ausgabe
- Sehr hohe Ein-/Ausgabebandbreite
- Beispiel: IBM OS/390, z/OS

Mainframe-Betriebssysteme

- Drei Arten der Prozessverwaltung:
- Batch-Verfahren/Stapelverarbeitung: Erledigung von Routineaufgaben ohne Benutzerinteraktion (Schadensmeldungen, Verkaufsberichte)
- Transaktionsverfahren/Dialogverarbeitung:
  Große Anzahl kleiner Aufgaben von vielen Nutzern (Überweisungen, Flugbuchungen)
- Timesharing: Quasi-parallele Durchführung vieler Aufgaben durch mehrere Benutzer (Anfragen an zentrale Datenbank)

Arten von Betriebssystemen (2)
Server-Betriebssysteme

Betriebssysteme für große PCs, Workstations oder auch Großrechner

Einsatz: z.B. Internetanbieter (Moodle, etc)
- Viele Benutzer gleichzeitig über Netzwerk bedienen
- Zuteilung von Hard- und Softwareressourcen

Beispiele: NetBSD (Unix), Windows Server

Arten von Betriebssystemen (3)
PC-Betriebssysteme

- Betriebssysteme für Personalcomputer
  Meist nur 1 Benutzer (oder wenige über Netzwerk)
- Einsatz: Programmierung, Textverarbeitung, Spiele, Internetzugriff, ...
  - Mehrere Programme pro Benutzer quasi-parallel
  - Aufteilung der Prozesse auf vorhandene Kerne
  - Zuteilung der Systemressourcen
- Beispiele: Linux, Windows, Mac OS X

Arten von Betriebssystemen (4)
Echtzeit-Betriebssysteme

- Einhalten **harter Zeitbedingungen** (vs. im Durchschnitt schnell)
- Einsatz: z.B. Betriebssysteme zur Steuerung maschineller Fertigungsanlagen (z.B. Autos)
- Aktion in einem fest vorgegebenen Zeitintervall (in jedem Fall, garantierte Deadlines)

Beispiele: VxWorks, OSEK-OS

Arten von Betriebssystemen (5)
Betriebssysteme für Eingebettete Systeme

- Eingebettete Systeme = „Computer, die man nicht unmittelbar sieht“
  - Einsatz: Fernseher, Mikrowelle, Mobiltelefon, Auto, ...
- Meist Echtzeitanforderungen
  - Wenig Ressourcen:
  - Kleiner Arbeitsspeicher
  - Geringer Stromverbrauch
- Beispiele: QNX, Windows CE, Windows Phone, iOS, Android

Architektur von Betriebssystemen Hauptkomponenten

- Meist gibt es eine **privilegierte Kontrollinstanz:** Zur Realisierung von Mehrnutzer- und/oder Mehrprozessbetrieb muss das Betriebssystem die Steuerung wichtiger Vorgänge übernehmen. Dadurch ergibt sich eine gewisse Strukturierung der zu realisierenden Aufgaben.
- **Typische Funktionskomplexe universeller Betriebssysteme:**
  - Kommunikation mit der Umgebung
  - Auftragsverwaltung
  - Benutzerverwaltung
  - Prozessverwaltung und -koordinierung
  - Betriebsmittelverwaltung
  - (Haupt-) Speicherverwaltung
  - Eingabe / Ausgabe Steuerung
  - Dateiverwaltung
- In Betriebssystemen für spezielle Einsatzbereiche sind einige Komponenten besonders ausgeprägt oder fehlen teilweise.

Systemsoftware

- Als Systemsoftware werden alle Programme zusammengefasst, die eine effiziente und komfortable Benutzung eines Computers ermöglichen. Neben dem Betriebssystem gehören dazu ergänzende, hardwareunabhängige Dienst- und Hilfsprogramme.
- Systemsoftware umfasst u. a.:
  - **Programmierumgebungen** (integrierte Software-Entwicklungssysteme) mit Editoren, Compiler bzw. Interpreter, Linker und Lader, Testhilfen (Debugger) sowie entsprechenden Bibliotheken und weiteren Verwaltungsprogrammen.
  - **Dienstprogramme** (tools, Utilities) für typische Arbeitsvorgänge (z. B. Suchen, Kopieren oder Sortieren von Daten) sowie zur Installation/Konfigurierung und Administration. Vielfach werden dazu heute auch Browser eingesetzt.
  - Programme zur Realisierung spezieller Betriebsformen.

Architekturmodelle

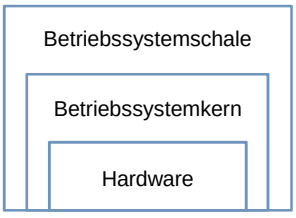
- Architekturmodelle verdeutlichen die Anordnung der Komponenten und deren funktionales Zusammenwirken. Bekannte Modelle sind:
- **Monolithische Architektur:** Alle wesentlichen Komponenten des Systems sind zu einem "homogenen Gebilde" zusammengefügt, das zwar u. U. effizient, aber nicht flexibel anpassbar ist.

Kern-Schale-Architektur

Das System besteht aus dem *privilegierten Kern (kernel),* der die wichtigsten Komponenten vereint (z.B. Prozessverwaltung), und

einer ***Schale (shell)*** für ergänzende Bestandteile (z.B. Kommando-Interpreter).

Typischer Vertreter dieses Modells ist UNIX / Linux.

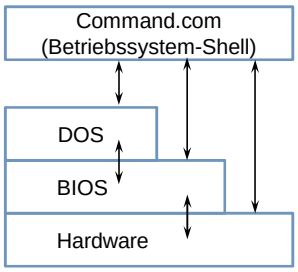


Hierarchische Schichten (Mehrschichtenmodell)

Das System wird modularisiert und in einzelne Schichten geteilt. Zwischen diesen sind ***Schnitt-stellen*** definiert, so dass sie austauschbar sind. Sie können mit abgestuften Privilegien ausgestattet sein.

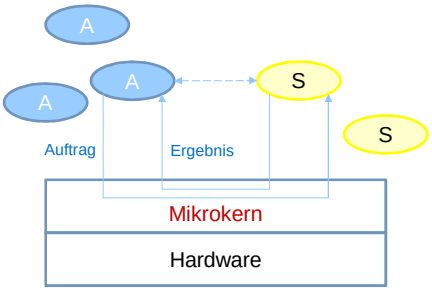
Diese Architektur ist weit verbreitet.

z. B. bei **MS-DOS**, OS/2, OpenVMS.



Mikrokern (*micro kernel*) (1)

Er bildet nur noch eine Art Infrastruktur **mit minimalem Funktionsumfang**. Alle anderen Betriebssystemfunktionen werden **durch Systemprozesse** außerhalb des Kerns erbracht, die flexibel modifiziert oder erweitert werden können.
Damit lassen sich günstig Client-Server-Modelle realisieren.
Mikrokerne findet man z.B. bei MACH, CHORUS, QNX/Neutrino, z.T. bei **MS Windows NT.**



A Anwendungsprozess
S Dienstleistungsprozess

Mikrokern (*micro kernel*) (2)

- **Mikrokerne** enthalten nur noch **elementare Funktionen**, z. B. zur Speicherverwaltung, IPC, Prozessverwaltung, Scheduling-Mechanismus, sowie einige hardwarenahe E/A-Funktionen.
- **Systemprozesse (Dienstleistungsprozesse)** realisieren dagegen z. B. die Dateiverwaltung.
- Mikrokerne findet man z.B. bei MACH, CHORUS, QNX/Neutrino, z.T. bei MS Windows NT.

Virtuelle Maschinen

- Als **virtuelle Maschine** (kurz **VM**) wird die Kapselung eines Rechnersystems innerhalb eines anderen bezeichnet.
- Die virtuelle Maschine bildet die Rechnerarchitektur eines real in Hardware existierenden oder hypothetischen Rechners nach.
- Die abstrahierende Schicht zwischen realem oder Host- (Gastgeber-) Rechner, auf dem die virtuelle Maschine ausgeführt wird, und der virtuellen Maschine wird *Hypervisor* oder **Virtual Machine Monitor** genannt.
- Der Hypervisor erlaubt in der Regel den Betrieb mehrerer virtueller Maschinen gleichzeitig auf einem physischen Rechner.

Typen virtueller Maschinen (1)
Systembasierte virtuelle Maschinen …

- … bilden einen Rechner so vollständig nach, dass Betriebssysteme, die für den realen Rechner entworfen wurden, sich auf der virtuellen Maschine genauso wie auf dem entsprechenden realen Rechner ausführen lassen.

  Beispiele sind Oracles **VirtualBox** oder VMwares **vSphere** oder

  - für Mainframes z.B. das IBM System/370 das diese Funktionalität 1972 einführte. Seitdem ist Virtualisierung ein Bestandteil aller Nachfolger-Systeme (alle modernen Systeme, wie das System z sind voll rückwärtskompatibel zu den Serie-S/360 Großrechnern der 1960er Jahre).

Typen virtueller Maschinen (2)
Prozessbasierte virtuelle Maschinen …

- *Begann mit dem Aufsatz Transportability of Software Applications on Microcomputers* von W. Wellbourne (1983) und der vorausgegangenen Arbeit *A Comparison of Pascal Intermediate Languages* von P. Nelson (1979).
- Ziel ist: Anwendungscode auf verschiedenen Rechnerarchitekturen ohne Änderungen auszuführen.
- Stellen dafür eine Laufzeitumgebung bereit.
- Bekannte Beispiele solcher Umgebungen mit entsprechenden virtuellen Maschinen sind die **Java-Laufzeitumgebung als Teil der Java-Plattform** und die Common Language Runtime als Teil des .NET Frameworks.

Nutzung der Dienste eines Betriebssystems …

- erfolgt im Allgemeinen entweder über die Benutzungsschnittstelle (das **User Interface, den Kommando-Interpreter**) oder über die **Programmierschnittstelle (API).**
- Interaktionsmöglichkeiten für den Kommando-Interpreter:
  - **Kommandosprachen:** Aufträge (Kommandos) werden durch Texteingaben hinter einem Bereitschaftszeichen (prompt) ausgelöst.
  - ***Text-Menüs, Bildschirm-Masken*** erlauben eine gute Bedienerführung mit strengen Eingabekontrollen.
  - **Grafische Benutzungsoberfläche** (Graphical User Interface, GUI): Die Bedienung erfolgt einfach mittels grafischer Eingabegeräte, Ausgaben erscheinen z. B. in Bildschirmfenstern (Windows).

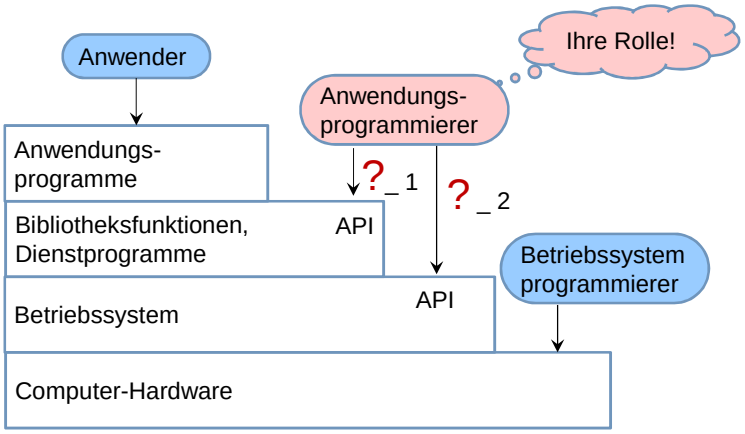
Programmierschnittstelle für Systemfunktionen

- Die **Programmierschnittstelle** (Application Programming Interface, **API**) definiert in ihrer Syntax und Semantik die Funktionen des Betriebssystems in Form von Systemdiensten (*system services*).
- Diese Systemdienste stehen für den Programmierer einer Anwendung im Allgemeinen in Funktionsbibliotheken (z. B. DLLs bei MS Windows) für die jeweils benutzte Programmierumgebung bereit und sind z. T. standardisiert (POSIX).
- Der Aufruf von Systemdiensten erfolgt unter **Nutzung spezieller Unterbrechungsmechanismen bzw. privilegierter Befehle** (Trap, Supervisor Call SVC) und sind Bestandteil des hardwareabhängigen Application Binary Interface (ABI).

Eine erste Zusammenfassung zu Betriebssystemen

- Betriebssystem = Software (also ein Programm)
- Abstrakte Schnittstelle zum Rechner
- Verwaltet Systemressourcen
- Historische Entwicklung in mehreren Stadien
- Verschiedene Arten von Betriebssystemen aufgrund verschiedener Anforderungen in unterschiedlichen Anwendungsgebieten
- Moderne Betriebssysteme sind Timesharing-Systeme mit Mehrprogrammbetrieb (plus zusätzliche Eigenschaften).
- Es gibt diverse Basis-Architekturen: Kern-Schale-Architektur, Hierarchische Schichten, Mikrokern (*micro kernel*), Virtuelle Maschinen

Schichtenarchitektur



Das Modul sys

- stellt Informationen in Konstanten, Funktionen und Methoden über den **Python-Interpreter** zur Verfügung
- Die **help-Funktion**, also `help(sys)`, liefert wertvolle Detailinformationen.
- Hier sollten Sie sich eine Übersicht verschaffen!

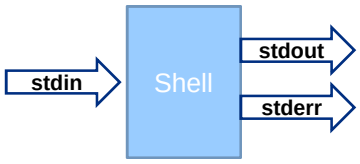
Modul sys - Beipiele

```
>>> import sys
>>> sys.version
'3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:57:36) [MSC v.1900 64 bit (AMD64)]'
>>> sys.version_info
sys.version_info(major=3, minor=6, micro=2, releaselevel='final', serial=0)
>>> sys.getrecursionlimit()
1000
>>> a = 1.0
>>> sys.getsizeof(a)
24
>>> a = 1
>>> sys.getsizeof(a)
28
>>> a = '1'
>>> sys.getsizeof(a)
50
```

Standard-Datenströme

Als Standard-Datenströme (*standard streams*) in Unix bzw. Linux sind **drei Datenströme** für die Ein- und Ausgabe definiert.

Viele Shell-Kommandos verwenden automatisch die Standardein- bzw. Standardausgabe, sofern keine Datei in der Kommandozeile für die Ein- oder Ausgabe angegeben worden sind.



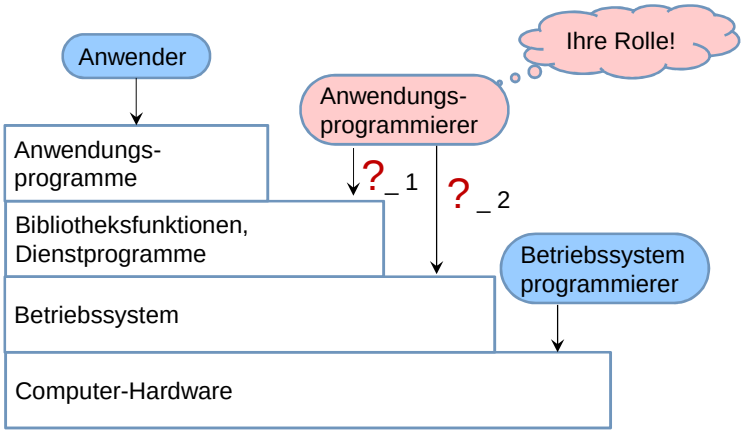
Als **stdin** gilt normalerweise die Tastatur.
stdout ist normalerweise das Terminalfenster / die Textkonsole.
Fehlermeldungen werden in **stderror** ausgegeben (üblicherweise auch das aktuelle Terminalfenster / die Textkonsole.

Umsetzen von stdin, stdout, stderr

- In Python gibt es die Möglichkeit auf die Standardkanäle, also `stdin, stdout und stderr`, schreibend und lesend zuzugreifen.
- Beispielsweise kann die Standardausgabe (stdout) in eine Datei umgeleitet werden, um Ausgaben zu archivieren oder sie offline weiterverarbeiten zu können.

```
>>> import sys
>>> print("Versuch")
Versuch
>>> save_stdout = sys.stdout
>>> testfile = open("test.txt","w")
>>> sys.stdout = testfile
>>> print("This text goes to test.txt.")
>>> sys.stdout = save_stdout
>>> testfile.close()
>>> print("Versuch")
Versuch
```

Schichtenarchitektur



Das os-Modul

Das os-Modul ist das wichtigste Modul zur Interaktion mit dem Betriebssystem

Ermöglicht durch **abstrakte Methoden** ein plattformunabhängiges Programmieren (in vielen Fällen!)

mit Funktionen wie os.system() oder der exec*()-Funktionsfamile ist es möglich, betriebssystemabhängige Programmteile einzubauen.

(Die Familie der exec*()-Funktionen besprechen wir später-)

Das os-Modul bietet viele unterschiedliche Methoden für den **Zugriff auf das Dateisystem.**

Unbedingt einmal `help(os)` ausführen. Bietet **sehr viele Funktionen**.

Beispiele: Zugriff aufs Dateisystem

Funktion	Beschreibung
`chdir(path)`	Das aktuelle Arbeitsverzeichnis (working directory) wird in path geändert.
`getcwd()`	Liefert einen String mit dem aktuellen Arbeitsverzeichnis zurück.
`listdir(path)`	Eine Liste des Inhaltes (Dateinamen, Unterverzeichnisse usw.) des Verzeichnisses path
`mkdir(path [, mode=0755])`	Ein neues Verzeichnis wird erzeugt. Falls übergeordnete Verzeichnisse in path nicht existieren, wird ein Fehler erzeugt und kein Verzeichnis angelegt.
`makedirs(name [, mode=511])`	Wie mkdir, legt jedoch auch übergeordnete Verzeichnisse automatisch an.
`renames(old, new)`	Wie rename, aber auch übergeordnete Verzeichnisse in "old" werden, falls nicht vorhanden angelegt. Allerdings nur, wenn die nötige Berechtigung vorliegt
rmdir(path)	Ein Verzeichnis wird gelöscht, falls es leer ist

Weitere Module für den Zugriff aufs Betriebssystem

- Weitere Funktionen, die auf Dateien und Verzeichnissen arbeiten, liefert das **Modul shutil**. Es bietet unter anderem Möglichkeiten Dateien und Verzeichnisse zu kopieren, z.B. shutil.copyfile(src,dst).
- **Modul platform** gibt infos zur zugrundeliegenden Platform:

  `machine(),node(), processors(), …`

Zusammenfassung

- Betriebssysteme sind heute sehr leistungsfähig.
- … aber man muss sie gut kennen!
- Wirkliche Plattformunabhängig ist schwierig zu erreichen.

Ausblick

Die Komplikationen mit mehreren Prozessen

Die exec-Familie

Interrupts und Exceptions

und, herzlichen Dank für Ihre Aufmerksamkeit!