
Einführung in die Informatik

Betriebssystem-Architekturen, Nebenläufigkeit und Netzwerke

Meik Teßmer

Inhalt

Lernziele

- Betriebssystem-Architekturen
- leichtgewichtige Alternative zu Prozessen: Threads
- Nebenläufigkeit und Synchronisation von Prozessen
- Netzwerke

Wiederholung: Betriebssysteme

- Multiprogramming: effiziente Ressourcennutzung
- Timesharing: faire Ressourcenverteilung
- Spooling: flexible Jobverwaltung
- Kernel-/User-Mode: Schutz der Prozesse
- virtueller Speicher: große Programme
- Prozesse und -verwaltung: Zustände, Scheduler
- Prozesskommunikation: Signale, Pipes/Dateisystem
- besondere Vorkehrungen bei Mehrbenutzersystemen

Architekturen

Aufbau von Betriebssystemen

- Monolith
- geschichtete Systeme
- virtuelle Maschinen
- Client-Server/Mikrokern
- verteilte Systeme

Monolith

- alle Systemaufrufe bzw. die umsetzenden Prozeduren sind als große Masse implementiert
- jede Prozedur kann jede anderen aufrufen
 - kein Information Hiding-Prinzip, keine Modularisierung
- minimale Struktur nur durch TRAP-Befehl (Umschalten von Kernel- und User-Mode)
- Beispiel: MS-DOS, erste MS Windows-Versionen

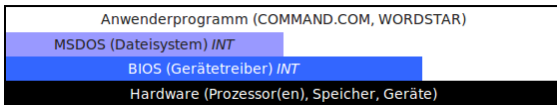


Abbildung 1: MS DOS

Geschichtete Systeme

erstes System: THE-System von Dijkstra (1968):

- unterste Schicht verwaltete Prozessor und Prozesse (Basis für Multiprogramming)
- nächste Schicht verwaltete Speicher
- nächsten Schichten setzten Prozesskommunikation, Ein-/Ausgabeverwaltung um
- letzten beiden Schichten: Benutzerprogramme und der Benutzer selbst

Verallgemeinerung dieses Konzepts: Ringe in MULTICS
(Unix-Vorläufer)

Geschichtete Systeme

die meisten modernen Betriebssysteme sind schichtartig konzipiert

Beispiel: GNU/Linux

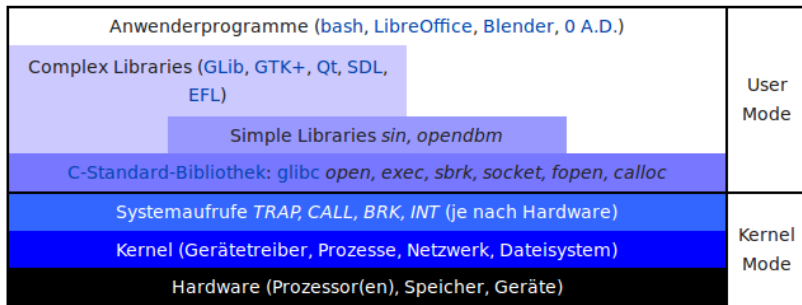


Abbildung 2: GNU/Linux

Virtuelle Maschinen

- zentrales Element: *Monitor*-Programm (auch Hypervisor genannt)
- läuft direkt auf der Hardware und stellt mehrere virtuelle Maschinen bereit
- virtuelle Maschine: exakte Kopie der unterliegenden Hardware (besitzen ebenfalls einen Kernel- und User-Mode, I/O, TRAP-Befehle etc.)
 - Betriebssysteme/Programme merken keinen Unterschied
- Beispiele: IBM z/VM, VMware ESX, Qemu/KVM, VirtualBox

Virtuelle Maschinen

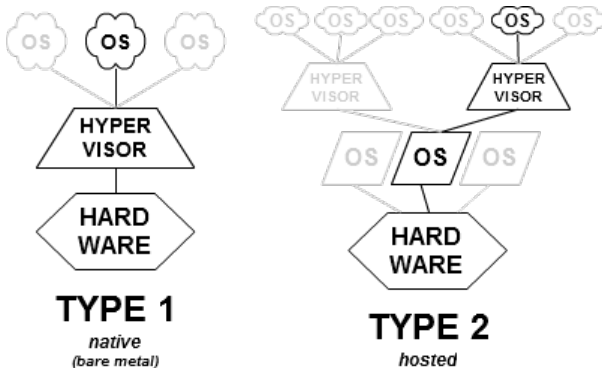


Abbildung 3: Virtuelle Maschinen: Varianten

Client-Server-Systeme

- Idee: Kernel so klein wie möglich
- *Mikrokernel*: verwaltet Kommunikation von Client- und Server-Prozessen
- Server-Beispiele: Speicherserver, Prozess-Server
- Vorteile:
 - Fehler bringen nicht das gesamte System zum Absturz
 - Konzept kann leicht auf verteilte Systeme erweitert werden
- Beispiele
 - Mikrokernel: Mach, L4 (Fiasco, Pistachio, Hazelnut)
 - Systeme: Minix, QNX, GNU/Hurd, L4Linux

Client-Server-Systeme

Vergleich Monolith und Client-Server

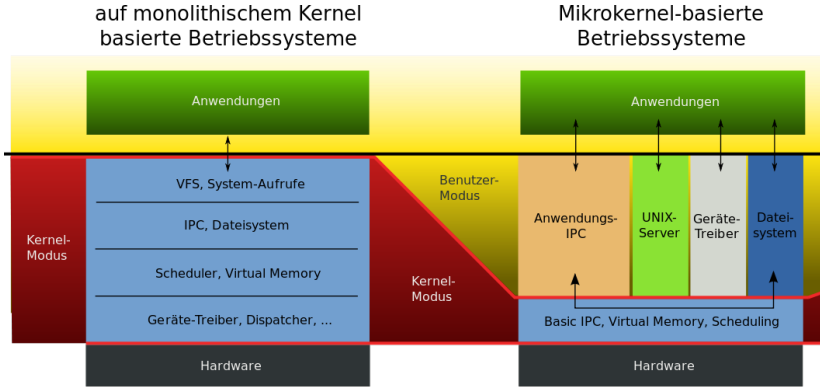
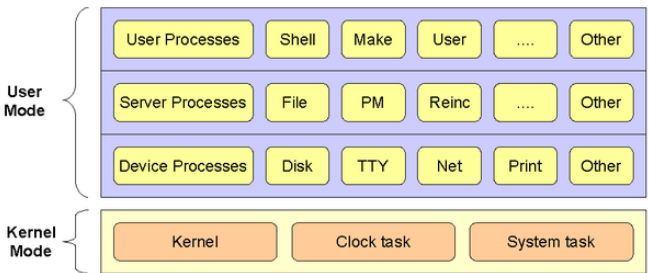


Abbildung 4: Monolith vs. Mikrokern

Client-Server-Systeme

Beispiel: MINIX3



The MINIX 3 Microkernel Architecture

Abbildung 5: MINIX3

Verteilte Systeme

- Client- und Serverprozesse können auf unterschiedlichen Maschinen laufen
- Kommunikation findet via Netzwerk statt
- Vorteil: Zusammenfassung von Rechenleistung und Speicher
- Beispiel-System: Plan 9, QNX

Variante: Cluster-Betriebssysteme wie MOSIX, OpenSSI

- Single-System Image: Rechnerverbund erscheint als *ein Rechner*

Nebenläufigkeit

Parallele Ausführung von Programmen

Was heißt das?

- Multiprogramming: mehrere Prozesse laufen „gleichzeitig“
 - Problem: Kontext-Switch kostet Zeit und Ressourcen
 - Prozess-Metadaten müssen gesichert werden (Stack, Program Counter etc.)
 - Metadaten eines anderen Prozesses müssen restauriert werden
- möglichst vermeiden oder so schnell wie möglich machen
- Alternative: Nebenläufigkeit *in* einem Prozess

Kleine Prozesse: Threads

- Ziel: schlanke parallele Ausführung von Code (Prozesse sind *groß* (Stack, Pointer usw.)
→ *Threads*
- Threads laufen *in* einem Prozess
→ jedes Programm hat also mindestens 1 *Thread*, den „Hauptstrang“
- Threads können wie Prozesse auch auf CPUs verteilt werden
- Problem: Zugriff auf gemeinsame Datenstrukturen
→ Locking, Semaphore usw.

Threads: Beispiel

Aufgabe: URLs aus dem Netz ziehen

```
from urllib.request import urlopen

output = open("das_internet.txt", "w")
urls = ['http://www.google.com',
        'http://www.facebook.com']
for url in urls:
    try:
        data = urlopen(url)
    except urllib.URLError as e:
        print('URL %s failed: %s' % (url, e))

    output.write(data.read().decode("latin1"))
output.close()
```

Threads: Beispiel

- Bausteine:
 - Thread-Erzeugung
 - Threads starten
 - auf Ende warten
- Erzeugung: Ableiten von Thread:

```
class FetchUrls(threading.Thread):  
    def __init__(self, urls, output):  
        threading.Thread.__init__(self)  
        ...
```

```
output = open('output.txt', 'w+')
```

```
thread1 = FetchUrls(urls1, f)
```

```
thread2 = FetchUrls(urls2, f)
```

Threads: Beispiel

- Thread starten:

```
thread1.start() # ruft run()-Methode  
thread2.start() # der Klasse auf
```

- auf Ende warten:

```
thread1.join() # Hauptstrang wartet  
thread2.join() # auf Threads
```

run()-Methode der Klasse

```
def run(self):
    print("Thread ID:", self.ident)
    while self.urls:
        url = self.urls.pop()
        try:
            d = urlopen(url)
        except urllib.URLLError as e:
            print('URL %s failed: %s' % (url, e))

    self.output.write(d.read().decode("latin-1"))
```

Threads: Zugriff auf gemeinsame Ressourcen

Problem: Ausgabe-Datei

- beide Threads schreiben, wenn sie eine URL herunter geladen haben
- Geschwindigkeit ist abhängig von Netzwerk, Antwortzeit und Größe des Inhalts
 - Ausgabe-Datei ist durchmischt
- einfache Lösung: Lock

Threads mit Locking

- Lock holen: `acquire()`
- Lock freigeben: `release()`
 - Synchronisation beider Threads bei Zugriff auf gemeinsame Ressourcen
- Anpassung des Codes:
 - 1 lock-Variable zur Klasse hinzufügen
 - 2 `run()`-Methode um `acquire()` und `release()` ergänzen

Threads mit Locking

```
class FetchUrls(threading.Thread):
    def __init__(self, urls, output, lock):
        threading.Thread.__init__(self)
        self.urls = urls
        self.output = output
        self.lock = lock # ← neu

    def run(self):
        while self.urls:
            url = self.urls.pop()
            ...
            self.lock.acquire() # ← neu
            self.output.write(d.read().decode("latin-1"))
            self.lock.release() # ← neu
```

Netzwerke

Internet

Das Internet

Für einige Neuland... für uns?

Anfänge des Datenaustauschs

- Datenaustausch früher: Disketten auf dem Schulhof, per Post, Ausdrucken/Abschreiben
- moderne Variante: USB-Sticks, CDs/DVDs
- beides nennt man auch *Sneakernet*
- *Bandbreite* recht gering (Diskette: 1,44 MB, DVD: 4,7 GB)
Übertragungs-Medium bestimmte die Bandbreite

Erste Verbindung von Rechnern

- Grundidee:
 - auszutauschende Daten werden in kleine Pakete zerlegt
 - Pakete werden separat übertragen
 - Empfänger setzt Pakete wieder zusammen

- 1969: ARPANET
 - Grundlage: Telefontechnik
 - Leitungen: 50 kbit/s
 - verband Forschungseinrichtungen und Universitäten
 - ab 1968: *Packet Switching* als Basistechnologie

ARPANET 1977

ARPANET LOGICAL MAP, MARCH 1977

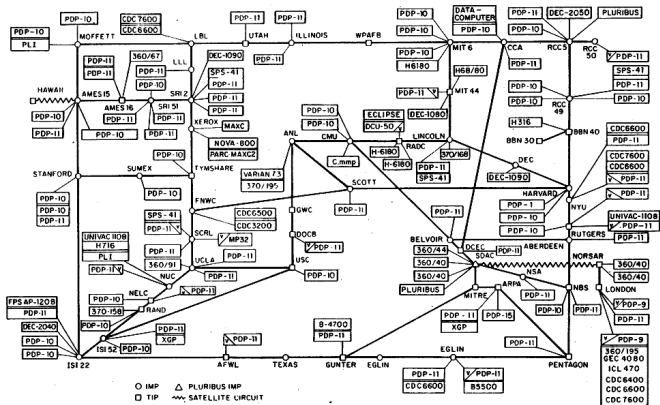


Abbildung 6: ARPANET

Netzwerk-Arten

- PAN: Personal Area Network
 - verbindet Geräte in direktem Umfeld (Telefon, Tablet, Headset, Uhr)
 - Reichweite: ca. 10m
 - Technik: WLAN, NFC, Bluetooth

- LAN: Local Area Network
 - verbindet Geräte in einem Gebäude
 - Technik: Ethernet (Kabel), WLAN

- WAN: Wide Area Network
 - Städte, Länder, Kontinente
 - Technik: Glasfaser, Satellit, Telefonleitungen

Mittlerweile wird Glasfaser auch im LAN-Bereich verwendet.

OSI-Referenzmodell

- OSI: Open Systems Interconnect
 - Entwicklung begann 1977
 - Standardisierung seit 1984

- Idee: Kommunikation wird auf 7 Schichten abstrahiert

Nr.	Schicht (eng.)	Schicht	Protokolle						
7	Application	Anwendung	HTTP		SMTP			FTP	
6	Presentation	Darstellung						Postscript	
5	Session	Sitzung				DNS		LDAP	
4	Transport	Transport	TCP					UDP	
3	Network	Vermittlung	IP				IPX		
2	Data Link	Sicherung		Ethernet	ATM		FDDI	ARP	
1	Physical	Bitübertragung							

Abbildung 7: OSI-Referenzmodell

TCP/IP-Stack

Basis der Internet-Kommunikation

OSI-Schicht	TCP/IP-Schicht	Beispiel
Anwendungen (7)	Anwendungen	HTTP , UDS , FTP , SMTP , POP , Telnet , OPC UA
Darstellung (6)		
Sitzung (5)		
		SOCKS
Transport (4)	Transport	TCP , UDP , SCTP
Vermittlung (3)	Internet	IP (IPv4, IPv6) , ICMP (über IP)
Sicherung (2)	Netzzugang	Ethernet , Token Bus , Token Ring , FDDI , IPoAC
Bitübertragung (1)		

Abbildung 8: TCP/IP-Stack

Kommunikation im Netz

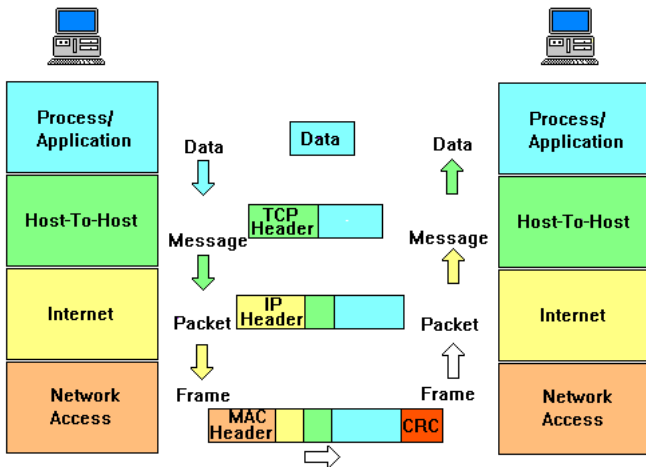


Abbildung 9: TCP-Stack

Netzklassen

- IP-Adressen: 32 Bit lange Zahl
- Darstellung: 4 Bytes in der Form a.b.c.d
- Adresse hat Host- und Netzwerkanteil

→ Einteilung der Adressen in einzelne Netze

	Klasse A-Netz	Klasse B-Netz	Klasse C-Netz
Netz-ID	8 Bit = 1 Byte	16 Bit = 2 Byte	24 Bit = 3 Byte
Host-ID	24 Bit = 3 Byte	16 Bit = 2 Byte	8 Bit = 1 Byte
Netzmaske	255.0.0.0	255.255.0.0	255.255.255.0
Adressklassen-ID (= Feste Bits im 1. Byte, 1. Quad)	0	10	110
Wertebereich (theoretisch)	0.0.0.0 bis 127.255.255.255	128.0.0.0 bis 191.255.255.255	192.0.0.0 bis 223.255.255.255
Anzahl der Netze	128 (= 2^7)	16384 (= $2^6 * 256$ = $64 * 256$)	2097152 (= $2^5 * 256 * 256$ = $32 * 256 * 256$)
Anzahl der Rechner im Netz	16777216 (= 256^3)	65536 (= 256^2)	256 (= 256^1)

Abbildung 10: Netzklassen

Netzklassen

seit 1993: Übergang zum *Classless Inter-Domain Routing* (CIDR)
→ Netzzugehörigkeit wird durch die Netzwerkanteil definiert

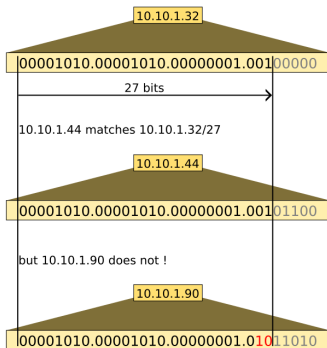


Abbildung 11: CIDR

Reservierte Bereiche

- dienen dazu, lokale Netze ohne Außenanbindung zu betreiben
→ Pakete aus diesen Netzen werden nicht weitergeleitet
- 10.0.0.0 - 10.255.255.255
- 172.16.0.0 - 172.31.255.255
- 192.168.0.0 - 192.168.255.255
- automatische Adressierung: 169.254.0.0 - 169.254.255.255

Welche Problem könnte nun angesichts der zunehmenden Verbreitung von netzwerkfähigen Geräten auftreten?

Abhilfe

■ NAT (Network Address Translation)

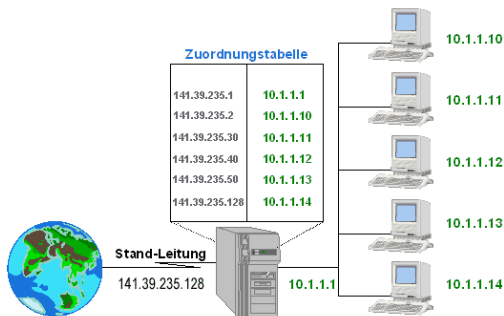


Abbildung 12: image

■ IPv6: 128 Bit lange Adressen ($2^{128} = \text{ca. } 340 \text{ Sextillionen}$)

Dienste im Internet heute

- Kommunikation: Instant Messing (WhatsApp, Signal, Matrix usw.), Kurznachrichtendienste (Mastodon, X etc.)
- Video-Anrufe: s.o.
- Informationsbeschaffung (Wikipedia, Verlage, arXiv)
- Streaming von Audio und Video
- „Cloud“: Remote-Datenspeicherung aller Art, Backup
- Teilen von Daten (Sync-and-Share wie OneDrive, Google Drive, iCloud usw.)
- Remote-Steuerung von Geräten (OP-Roboter, Industrieanlagen) in Echtzeit
- Zugriff auf KI-Dienste

Folgen?

Ende

Quellen:

- Tanenbaum: Moderne Betriebssysteme, 2. Aufl.
- Abbildungen: Wikipedia