Inhalt

1 LZ_01_Überblick

1.1 Die Shell

Durch Befehle in der Shell kann man auf alle Betriebssystemobjekte zugreifen

1.1.1 Verarbeitungszyklus der Shell

1. Warten auf Eingabe einer Zeile
2. Interpretiere die Zeile (Kommando + Parameter)
3. Ausführen
4. Zurück zu Schritt 1

1.1.2 Kommandos

touch <Datei>	Erstellt eine Datei
mkdir <Verzeichnis>	Erstellt ein Verzeichnis
pwd	Ausgabe working directory
cd <Verzeichnis>	Wechselt zum Verzeichnis
wc <Datei>	Zählt Wörter der Datei
man	Handbuch
grep “wort“ <Datei>	Sucht nach dem Wort in der Datei
cat	Zeigt Inhalt der Datei
less <Datei>	Ruft Textdateien auf

1.1.3 Besonderheiten der Shell

- Mehrere Kommandos auf einer Zeile mit ;: z.B.: echo Hallo; ls *.pdf
- Trennung mit && Ausführung nur wenn 1. Kommando erfolgreich
- ls -1 *.pdf > <Datei>: alle pdf Dateien umlenken auf einer Datei, -1 bedeutet Langform
- ls -l *.pdf | wc: über pipe verbinden, also ls ist die Eingabe von wc

1.1.4 Übung – Grep Befehle

3. Müllers ohne Doppelnamen
   - ⇨ grep -i “^mülle*r\> “ schausteller.txt
   - ⇨ -i: nicht case sensitive
   - ⇨ ^: wie es anfangen soll
   - ⇨ *: wie es enden soll
   - ⇨ \> : danach soll folgen, hier white space

2 LZ_02_Informationen

- Informationen: bilden Inhalt einer Nachricht
- Zeichen: Elemente der Kommunikation
- Glyphen: bezeichnet äußere Form des Zeichens

⇨ Zeichen müssen zur Kommunikation eine Bedeutung zugewiesen werden

2.1 Zahlensysteme

- Ziffer: Menge der Elemente von Symbolen
- Zahl: Anordnung von Ziffern, die in einem Zahlensystem einen bestimmten Wert besitzt

2.1.1 Klassifikation von Zahlensystemen

- Ziffernwertsystem: Zahlenwert durch die Ziffern bestimmt, Anordnung spielt keine Rolle wie beim römischen Zahlensystem
- Stellenwertsystem: Zahlenwert durch die Anordnung der Ziffern bestimmt

2.2 Polyadisches Zahlensystem KR!! – nur das

$(55)_{10}$ in Dual => $(110111)_2$

$n_0 = 55$	DIV 2 =	27	$a_0 = 55$	MOD 2 =	1
$n_1 = 27$	DIV 2 =	13	$a_1 = 27$	MOD 2 =	1
$n_2 = 13$	DIV 2 =	6	$a_2 = 13$	MOD 2 =	1
$n_3 = 6$	DIV 2 =	3	$a_3 = 6$	MOD 2 =	0
$n_4 = 3$	DIV 2 =	1	$a_3 = 3$	MOD 2 =	1
$n_5 = 1$	DIV 2 =	0	$a_3 = 1$	MOD 2 =	1

Von unten nach oben lesen!
$(110111)_2$

die **Hexadezimaldarstellung** $(ABCD)_{16}$ in Decimal

$(ABCD)_{16} = A \cdot 16^3 + B \cdot 16^2 + C \cdot 16^1 + D \cdot 16^0$

$(ABCD)_{16} = 10 \cdot 16^3 + 11 \cdot 16^2 + 12 \cdot 16^1 + 13 \cdot 16^0$

$(ABCD)_{16} = 40960 + 2816 + 192 + 13 = (43981)_{10}$

$$(1\,0101\,0101)_2 = \quad 0001\,|\,0101\,|\,0101$$
$$(1\,0101\,0101)_2 = \quad 1\;|\;5\;|\;5$$
$$(1\,0101\,0101)_2 = (155)_{16}$$

duale Darstellung des Dezimalbruchs $(0{,}125)_{10}$ => 1/8, p = 2 und r0 := g = 1 folgt.

$a_{-1} = 2^* 1$	DIV 8 =	0	$r_0 = 2^*1$	MOD 8 =	2
$a_{-2} = 2^* 2$	DIV 8 =	0	$r_1 = 2^*2$	MOD 8 =	4
$a_{-3} = 2^*4$	DIV 8 =	1	$r_2 = 2^*4$	MOD 8 =	0

Von oben nach unten lesen ab 0.[..]
$(0{,}001)_2$

Beispiel: Wir bestimmen die Darstellung $\left(\frac{a}{b}\right)_{10} = (0, 10\overline{01})_2$.

Basis $p = 2$, **Vorperiode** $s = 2$, **Periode** $t = 2$ und $b := p^{s+t} - p^s$. Damit folgt:

$$a = b \cdot (0, 10\overline{01})_2 = \left(p^{s+t} - p^s\right) \cdot (0, 10\overline{01})_2 = \left(2^{2+2} - 2^2\right) \cdot (0, 10\overline{01})_2$$
$$a = 2^4 \cdot (0, 10\overline{01})_2 - 2^2 \cdot (0, 10\overline{01})_2$$
$$a = (1001)_2 - (10)_2$$
$$a = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 - 1 \cdot 2^1 - 0 \cdot 2^0$$
$$a = 2^3 + 2^0 - 2^1 = 8 + 1 - 2 = 7$$

Mit $b := p^{s+t} - p^s = 2^4 - 2^2 = 16 - 4 = 12$ ergibt sich

$$(0, 10\overline{01})_2 = \left(\frac{7}{12}\right)_{10}$$

2.3 Konvertierung KR

$(n)_2$	$(n)_8$
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

$(n)_2$	$(n)_{16}$
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

2.3.1 zwischen Dual.- und Oktalsystem

Dreiergruppen bilden und jede dieser Gruppen einzeln in das Oktalsystem überführt.

Beispiel: $(110\,111\,001\,110\,010)_2 = (67162)_8$

110 | 111 | 001 | 110 | 010	Dualdarstellung
6 | 7 | 1 | 6 | 2	Oktaldarstellung

2.3.2 zwischen Dual.- und Hexadezimalsystem

Vierergruppen bilden und jede dieser Gruppen einzeln in das Hexadezimalsystem überführt.

Beispiel: $(1010\,1101\,1010)_2 = (\mathrm{A\ D\ A})_{16}$

1010 | 1101 | 1010	Dualdarstellung
A | D | A	Hexadezimaldarstellung

2.4 Das Horner Schema NKR

tabellarischer Form:

	a_5	a_4	a_3	a_2	a_1	a_0
$p=2$	$s\cdot p+a_5$	$t_5\cdot p+a_4$	$t_4\cdot p+a_3$	$t_3\cdot p+a_2$	$t_2\cdot p+a_1$	$t_1\cdot p+a_0$
$s=0$	$=t_5$	$=t_4$	$=t_3$	$=t_2$	$=t_1$	$=t_0$

Einsetzen der Ziffern $a_5=1, a_4=1, a_3=1, a_2=1, a_1=0, a_0=1$ und $p=2$ liefert:

	1	1	1	1	0	1
$p=2$	$0\cdot 2+1$	$1\cdot 2+1$	$3\cdot 2+1$	$7\cdot 2+1$	$15\cdot 2+0$	$30\cdot 2+1$
$s=0$	1	3	7	15	30	61

Lösung: $n = (11\,1101)_2 = (61)_{10}$

2.5 Rechenoperationen im Dualsystem NKR

2.5.1 Die Addition im Dualsystem

Beispiel: $(45)_{10} + (54)_{10}$

	0	1	0	1	1	0	1	$= (45)_{10}$
+	0	1	1	0	1	1	0	$= (54)_{10}$
Ü	1	1	1	1	0	0		
	1	1	0	0	0	1	1	$= (99)_{10}$

Das Rechenwerk eines Prozessors implementiert eine Addiereinheit, welche eine gewünschte Addition s = n1 + n2 ausführt.

$0+0 = 0\,\ddot{U}\,0$
$0+1 = 1\,\ddot{U}\,0$
$1+0 = 1\,\ddot{U}\,0$
$1+1 = 0\,\ddot{U}\,1$
$1+1+1\ (\text{vom } \ddot{U}) = 1\,\ddot{U}\,1$

2.5.2 Die Subtraktion im Dualsystem´

Beispiel: $(54)_{10} - (45)_{10}$

	0	1	1	0	1	1	0	$= (54)_{10}$
−	0	1	0	1	1	0	1	$= (45)_{10}$
Ü	0	0	1	0	0	1		
	0	0	0	1	0	0	1	$= (9)_{10}$

Für die Subtraktion verwendet der Prozessor i.A. ebenfalls die Addiereinheit des Rechenwerks.
Idee: d = n1 ≠ n2 = n1 + (≠n2).

$0-0 = 0\,\ddot{U}\,0$
$0-1 = 1\,\ddot{U}\,1$
$1-0 = 1\,\ddot{U}\,0$
$1-1 = 0\,\ddot{U}\,0$

2.5.3 Die Multiplikation im Dualsystem

Die Multiplikation von Dualzahlen erfolgt wie im Dezimalsystem und ist damit eine wiederholte Addition.

Beispiel: $(22)_{10} \cdot (22)_{10} = (484)_{10}$

	1	0	1	1	0	·	1	0	1	1	0
	1	0	1	1	0						
		0	0	0	0	0					
			1	0	1	1	0				
				1	0	1	1	0			
+					0	0	0	0	0		
0	0	1	1	1	1	0	0	0			
=	1	1	1	1	0	0	1	0	0		

Lösung: $(1\,1110\,0100)_2 = (484)_{10}$

2.5.4 Die Division im Dualsystem

Die Division von Dualzahlen erfolgt wie im Dezimalsystem und ist damit eine wiederholte Subtraktion.

Beispiel: $(105)_{10} : (7)_{10} = (15)_{10}$

	1	1	0	1	0	0	1	:	1	1	1	=	1	1	1	1
−		1	1	1	|	|	|								
		1	1	0	↓	|	|								
		1	1	0	0	|	|								
−			1	1	1	|	|								
			1	0	1	↓	|								
			1	0	1	0	|								
−				1	1	1	|								
					1	1	↓									
					1	1	1									
−					1	1	1									
					0	0	0									

Ergebnis: $(1111)_2 = (15)_{10}$

2.6 Relle Zahlen NKR

2.6.1 Darstellung von Festpunktzahlen

Bei den Festpunktzahlen steht das Trennzeichen (Komma oder Punkt) immer an einer bestimmten festgeschriebenen Stelle, wobei das Trennzeichen selbst nicht gespeichert wird.

$$11,011)_2 = 1 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3}$$
$$11,011)_2 = 2 + 1 + 0 \cdot 0{,}5 + 0{,}25 + 0{,}125$$
$$11,011)_2 = (3,375)_{10}$$

2.6.2 Gleitkommadarstellung

Jede reelle Zahl n kann als das Produkt einer Festpunktzahl multipliziert mit einem als Potenz geschriebenen Skalierungsfaktor dargestellt werden.

$$(n)_{10} = 123,0 \quad = 1{,}23 \cdot 10^2$$
$$(n)_{10} = 0,00012345 = 1{,}2345 \cdot 10^{-4}$$
$$(n)_{10} = 45003 \quad = 4{,}5003 \cdot 10^4$$

Es sind extrem kleine und auch extrem große Zahlen darstellbar.
Die Genauigkeit ist dabei relativ zur Größe der Zahl.

2.7 Codes zur Darstellung von Zeichen NKR

2.7.1 Codierung: Definitionen

Ein **Alphabet** ist eine endliche Menge X von Elementen
Ein **Element** (z. B. x1 œ X) wird als Symbol des Alphabets bezeichnet

2.7.2 Eigenschaften und Anwendungen von Codierungen

Eine Codierung ...

- ... legt fest, in welcher die Form Daten gespeichert werden,
- ... definiert die Zeichen einer Sprache, oder eines Alphabets,
- ... verschlüsselt Daten,
- ... erkennt Fehler in einer Datenübermittlung,
- ... steigert die Rechenleistung,

ASCII-Code = 2^8 = 256 Zeichen

Unicode = 2^{16} = 65536 Zeichen (später $17 \cdot 2^{16}$ = 1.114.112)

ASCII-Code reicht nicht => **Unicode (Die Erfassung und Katalogisierung aller Schriftzeichen und Symbole aus allen Kulturen.)!**

2.8 Grundlagen der Booleschen Algebra NKR

Eine **Menge** $\mathbb{M}$ in der die **Zustände** $\{0,1\}$, die **Operationen**

$\wedge : \mathbb{M} \times \mathbb{M} \longrightarrow \mathbb{M}$ **Konjunktion** (Und, AND)

$\vee : \mathbb{M} \times \mathbb{M} \longrightarrow \mathbb{M}$ **Disjunktion** (Oder, OR)

$\neg : \mathbb{M} \longrightarrow \mathbb{M}$ **Negation** (Nicht, NOT)

sowie für **alle** $x, y, z \in \mathbb{M}$ die **folgenden Gesetze** gelten

a)	**Kommutativgesetze**:	$x \wedge y = y \wedge x$	und	$x \vee y = y \vee x$
b)	**Assoziativgesetze**:	$(x \wedge y) \wedge z = x \wedge (y \wedge z)$	und	$(x \vee y) \vee z = x \vee (y \vee z)$
c)	**Distributivgesetze**:	$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$	und	$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$
d)	**Komplement**:	$x \wedge \neg x = 0$	und	$x \vee \neg x = 1$
e)	**Neutrales Element**:	$x \wedge 1 = x$	und	$x \vee 0 = x$
f)	**Eins-Element**:	$x \wedge 0 = 0$	und	$x \vee 1 = 1$

heißt eine Boolsche Algebra.

Weitere wichtige Rechenregeln

Für alle $x, y \in \mathbb{M} = \{0,1\}$ gelten die folgenden **Gesetze**:

g)	**Absorbtionsgesetze**:	$x \wedge (x \vee y) = x$	und	$x \vee (x \wedge y) = x$
h)	**Idempotenzgesetze**:	$x \wedge x = x$	und	$x \vee x = x$
i)	**Involutionsgesetz**:	$\neg(\neg x) = x$		
j)	**Gesetze von de Morgan**:	$\neg(x \wedge y) = \neg x \vee \neg y$	und	$\neg(x \vee y) = \neg x \wedge \neg y$
k)	**Auslöschungsregel I**:	$x \wedge \neg x = 0$	und	$x \vee \neg x = 1$
ℓ)	**Auslöschungsregel II**:	$x \wedge (y \vee \neg y) = x$	und	$x \vee (y \wedge \neg y) = x$

3 LZ_03_Rechneraufbau

3.1 Darstellung von Datenmengen (vll. NKR)

Halfbyte = 2^14 | **Byte** = 2^8 | **word** = 2^16 |**double** 2^32

Umrechnungsbeispiele

Beispiel 1: Wieviel **Bit** entsprechen **4 Gibibyte** (GiB)?

$4\,\text{GiB} = 4 \cdot 2^{10}\,\text{MiB} = 4 \cdot 2^{10} \cdot 2^{10}\,\text{KiB} = 4 \cdot 2^{10} \cdot 2^{10} \cdot 2^{10}\,\text{Byte}$

Mit $1\,\text{Byte} = 8\,\text{Bit}$ folgt.

$4\,\text{GiB} = 4 \cdot 2^{10} \cdot 2^{10} \cdot 2^{10} \cdot 8\,\text{Bit} = 2^2 \cdot 2^{30} \cdot 2^3\,\text{Bit} = 2^{35}\,\text{Bit} = 34.359.738.368\,\text{Bit}$

Beispiel 2: Wie viel sind **4 Megabyte** (MB) ausgedrückt in **Mebibyte** (Mib)?

$1\,\text{MB} = \alpha\,\text{MiB}$

$1 \cdot 10^6\,\text{Byte} = \alpha \cdot 2^{20}\,\text{Byte} \implies \alpha = \frac{10^6}{2^{20}}$

$1\,\text{MB} = \frac{10^6}{2^{20}}\,\text{MiB}$ daraus folgt:

$4\,\text{MB} = 4 \cdot 1\text{MB} = 4 \cdot \frac{10^6}{2^{20}}\,\text{MiB} = 3,8147\,\text{MiB}$

Dezimal-Präfixe 10^n (SI-Präfixe)				
1 Byte	(B)	=	10^0	Byte
1 Kilobyte	(kB)	=	10^3	Byte
1 Megabyte	(MB)	=	10^6	Byte
1 Gigabyte	(GB)	=	10^9	Byte
1 Terabyte	(TB)	=	10^{12}	Byte
1 Petabyte	(PB)	=	10^{15}	Byte
1 Exabyte	(EB)	=	10^{18}	Byte
1 Zettabyte	(ZB)	=	10^{21}	Byte
1 Yottabyte	(YB)	=	10^{24}	Byte

Binär-Präfixe 2^n (IEC-Präfixe)				
1 Byte	(B)	=	2^0	Byte
1 Kibibyte	(KiB)	=	2^{10}	Byte
1 Mebibyte	(MiB)	=	2^{20}	Byte
1 Gibibyte	(GiB)	=	2^{30}	Byte
1 Tebibyte	(TiB)	=	2^{40}	Byte
1 Pebibyte	(PiB)	=	2^{50}	Byte
1 Exbibyte	(EiB)	=	2^{60}	Byte
1 Zebibyte	(ZiB)	=	2^{70}	Byte
1 Yobibyte	(YiB)	=	2^{80}	Byte

EVA-Prinzip - Eingabe-Verarbeitung-Ausgabe

Ein Gerät nach reinem EVA-Prinzip ist zustandslos.

3.1.1 ECC- Error Correcting Code KR!!!

- Erkennt die Fehler im Ram und korrigiert sie.
- Bits im Arbeitsspeicher können kippen und somit werden falsche Werte ausgelesen.
  ⇨ Durch Paritätsrechnung und Korrektur kann dies erkannt und repariert werden.

3.2 Das Polling-Verfahren

CPU fragt im dauer loop nach einer Eingabe

- Hohe CPU-Belastung
- Es kann vorkommen, dass die CPU eine Komponente gerade abgefragt

3.3 Das Interrupt-Verfahren

Die Komponente meldet ein Interrupt-Signal an die CPU

- Es gibt kaum einen Zeitverzug
- Spezielle Hardware erforderlich

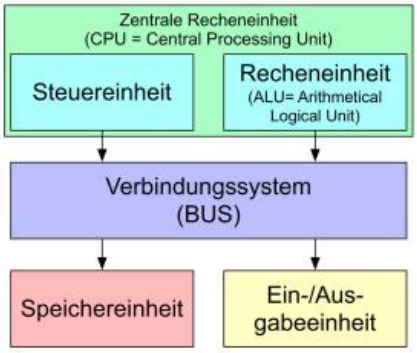
3.4 Von Neumann Architektur KR

- Architektur Konzept für einen speicherprogrammierbaren Universalrechner.
- Programm und Daten sind binär dargestellt und befinden sich in einem universellen Speicher

- **Zentrale Recheneinheit CPU**
  - **Recheneinheit (ALU)**
    führt Rechenoperationen und logische Verknüpfungen aus.
    Steuereinheit (Control Unit)
    interpretiert die Anweisung eines Programms und steuert die Befehlsabfolge.

- **Speichereinheit (memory)**
  speichert Programme und Daten, die für das Rechenwerk zugänglich sind.

- **Ein- Aufgabeeinheiten (I/O Unit)**
  - Steuert die Ein- und Aufgabe von Daten zum Anwender (Tastatur, Bildschirm) oder zu anderen Systemen (Schnittstellen).
- **Verbindungsbus**
  - Verbindet die Komponenten des Rechners untereinander.



3.5 Der von-Neumann-Zyklus KR

1. **FETCH**: (Befehlsholphase)
   - ⇨ Während der FETCH-Phase wird der nächste auszuführende Maschinenbefehl aus dem Speicher geholt
2. **DEOCODE:** (Dekodierphase)
   - ⇨ Geholte Maschinenbefehl Befehls-Decodier (ID) in Schaltinstruktionen für die Hardware umgewandelt
3. **FETCH OPERANDS:** (Operandenphase)
   - ⇨ Notwendige Operanden aus dem Speicher in die ALU geladen für Maschinenbefehl
4. **EXECUTE:** (Ausführphase)
   - ⇨ Maschinenbefehl von ALU ausgeführt
5. **UPDATE PROGRAM COUNTER:** (Updatephase)
   - ⇨ Program Counter (PC) wird erhöht, damit auf den nächsten Befehl zugegriffen werden kann

3.6 Merkmale der von Neumann Architektur

- Universeller Speicher für Operanden (Daten) und Maschinenbefehle (Programme)
- Programmsprünge => flexible Programmsteuerung
- Speicherverletzung: kein Schutz unberechtigter Zugang

=> Computer kann selbstständig logische Entscheidungen treffen

3.6.1 Harvard Architektur

- Wie von Neumann Architektur
- Außer Trennung von Programmcode und Datencode in separate Speicher & getrennte Busse für Programmcode und Datencode

3.6.2 Vorteile & Nachteile von Neumann & Harvard

- Vorteil:
  - ⇨ flexible Aufteilung des Speichers zwischen Programmen und Daten
  - ⇨ Programmsprünge möglich
- Nachteil:
  - ⇨ Programmcode und Datencode teilen sich ein Bussystem => langsamer
  - ⇨ Von Neumann Flaschenhals: Transfergeschwindigkeit der CPU durch internen Bus langsamer als die Verarbeitungsgeschwindigkeit der CPU => Bussystem bestimmt speed. Behebbar durch schnelle Cache Speicher in der CPU
  - ⇨ Programme und Daten können nicht im Speicher unterschieden werden
  - ⇨ Falsche Adressierung sehr schlechte Auswirkungen
- Bei Harvard umgetauscht bei Vor- und Nachteilen:
  - 2 Bussysteme => schneller & kein großes Problem bei falscher Adressierung, aber keine flexible Aufteilung von Speicher

3.7 Die Grundaufgaben des Betriebssystems KR

Programmsammlung: Sammlung verschiedener Programme, die den Betrieb ermöglichen.

3.7.1 Grundaufgaben des Betriebssystems

1. Unterstützung der Interaktion zwischen Mensch und Maschine, Schnittstellen:
   - ⇨ Bedienschnittstelle – User Interface (UI)
   - ⇨ Programmierschnittstelle – Application Programming Interface (API)
2. Verwaltung aller Komponenten eines komplexen Computersystems

3.7.2 Bedienschnittstelle

- 2 Arten der Kommunikation zwischen Anwender und Betriebssystem & Software
- Dialogorientierte Konsole (Terminal): dialogbasierte Kommunikation über Tastatur
- Benutzeroberfläche (Graphical User Interface - GUI): aktivieren von Schaltflächen über Maus

3.7.3 Programmierschnittstelle

- Kommunikationspartner zum Programmierer ist nicht die Maschine, sondern eine virtuelle Maschine = Betriebssystem => einfacher zu verstehen und zu programmieren
  - ⇨ Nimmt Komplexität versteckt wie Datei schreiben und abspeichern muss man nicht Schreib-/Lesekopf steuern und Speichergröße des Dokumentes wissen, übernimmt Betriebssystem

3.7.4 Weitere Aufhaben

- Verwaltung des Prozessors: Zuweisung von Programmen mit Ordnungsalgorithmus
- Verwaltung des Arbeitsspeichers:
- Verwaltung der Ein- und Ausgänge: ermöglicht und kontrolliert Vereinheitlichung der Programmzugriffe auf physikalische Ressourcen mit Hilfe von Treiber-Programmen (Kernelmodule)
- Verwaltung der Ausführung der Anwendungen: erforderliche Ressourcen zuteilen und Ablauf überwachen der Anwendungen
- Rechteverwaltung: garantiert Ressourcen für Anwender und Programme nur bei entsprechen Rechten möglich
- Dateiverwaltung: überwacht Lese- und Schreiboperationen im Dateisystem und verwaltet Zugriffsrechte auf Dateien
- Informationsverwaltung: Betriebssystem stellt Indikatoren zur Verfügung und ermöglich korrekte Funktionsweise des Computers zu diagnostizieren

3.8 Verbreitete Architekturkonzepte von Mikroprozessoren

3.8.1 Complex Instruction Set Computer (CISC)

- Prozessor mit großem Befehlssatz (Maschinenbefehle > 100)
  - ⇨ **Kleine** Anzahl von Registern
  - ⇨ Kann **komplexe** Operationen ausführen
  - ⇨ Unterstützt **viele** unterschiedliche Datentypen > 10
  - ⇨ Befehlsausführungen **unterschiedlich** lang
  - ⇨ Interpretiert Maschinenbefehle durch eigenen Microcode

3.8.2 Reduced Instruction Set Computer (RISC)

- Prozessor mit geringem Befehlssatz (Maschinenbefehle < 100)
  - ⇨ **Große** Anzahl von Registern
  - ⇨ Kann **einfache** Operationen ausführen
  - ⇨ Unterstützt **wenige** unterschiedliche Datentypen < 4
  - ⇨ Befehlsausführungen **gleich** lang
  - ⇨ Interpretiert Maschinenbefehle in eigener Hardware

3.8.3 Argumente von CISC und RISC

- CISC
  - ⇨ Komplexe Maschinenbefehle => geringer Speicherplatzverbrauch & Compileraufbau einfacher & weniger Zugriff auf Speicher => schneller ausführbar
- RISC
  - ⇨ Einheitlicher Aufbau & Befehlsausführung in einem CPU-Zyklus => Pipelining möglich
  - ⇨ Weniger Befehle => schnellere Decodierung

3.9 Pipelining KR

- Zur Steigerung der Geschwindigkeit der Befehlsbearbeitung wird bei RISC-Prozessoren
- unterschiedliche Phasen der Befehlsabarbeitung parallelisiert
- tn= tb + (n-1) * (tb/k) = **(n*k)/(n+k-1) || n= Anzahl der Befehle k= phasen**

3.9.1 Pipelining: Hazards

In der Praxis kommt es allerdings immer wieder zu Abhängigkeiten und Konflikten zwischen den Maschinenbefehlen

Die Data-Hazards (Datenabhängigkeiten)

- Zwischen zwei aufeinanderfolgenden Befehlen kommt es dann zu einer Datenabhängigkeit, wenn der zweite Befehl das Ergebnis des ersten Befehls benötigt, dieses aber noch nicht vorliegt, weil der erste Befehl noch nicht vollständig abgearbeitet ist

die Control-Hazards (Abhängigkeiten durch den Programmablauf)

- Ng

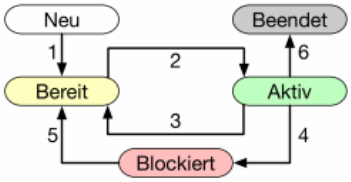
3.10 Programm und Prozess KR

- Ein **Programm** ist eine statische Beschreibung für einen Algorithmus
- Ein **Prozess** ist eine dynamische Ausführung eines Programms auf einem Computer.

Voraussetzung: (**minimale Prozessmodel**)

- Zu einem Zeitpunkt ist nur ein Prozess aktiv
- Es können weitere Prozesse im Status aktiv/blockiert sein

- Zu jedem **Zeitpunkt** ist immer nur **ein** Prozess **aktiv**.
- Es können **weitere** (mehrere) **Prozesse** im **Status Bereit** oder **Blockiert** sein.



Status	Beschreibung
Neu	Der Prozess wurde gerade erzeugt
Bereit	Der Prozess könnte laufen/rechnen, wenn die CPU frei wäre
Aktiv	Der Prozess wird ausgeführt/rechnet
Blockiert	Der Prozess wartet auf ein Ereignis (z. B. Benutzereingabe)
Beendet	Der Prozess ist fertig ausgeführt

3.10.1 Der Prozesskontrollblock IDK

Das Betriebssystem fasst alle zu einem Prozess gehörenden Informationen in einem Prozesskontrollblock zusammen.:

Prozessattribut	*Information*
Prozessnummer (PID)	*Systemweit eindeutige Nummer (Zahl)*
Prozessnummer des Elternprozesses (PPID)	*Prozess, der diesen Prozess erzeugt hat*
Benutzer- und Gruppenidentität (UID, GID)	*Besitzer- und Gruppenzuordnung des Prozesses*
Schedulingzustand und -priorität	*Aktueller Prozess-Status und seine Vorrangigkeit*
CPU-Register	*Stand des Programmzählers (PC), usw.*
Offene Dateien	*Liste der von diesem Prozess geöffneten Dateien inklusive der aktuellen Position in der Datei*
Aktuelles Arbeitsverzeichnis	*Das Verzeichnis in dem relative Dateinamen dieses Prozesses beginnen*
Ressourcenbeschränkungen	*Bezüglich Hauptspeicher, CPU-Zeit, Datei-Locks, usw.*
Terminal	*Terminalnummer von dem aus der Prozess gestartet wurde*

Der **Scheduler** (engl. scheduler - Zeitplaner):
wählt aus der Liste der Prozesse mit dem Status „Bereit“ einen Prozess aus, der als nächster in die CPU kommt (in den Status „Aktiv“ wechselt).

Der **Dispatcher** (engl. dispatcher; Zuteiler, Umschalter):
schaltet die CPU zwischen den einzelnen Prozessen hin und her.

3.11 Virtueller Speicher

- RAM gilt als Zwischenspeicher, dieser ist aber begrenzt => Lösung durch Memory Management Unit (MMU)
- Idee:
  - ⇨ Vortäuschen eines sehr großen Hauptspeichers
  - ⇨ Einblenden benötigten Speicherbereiche aus der HDD, Speicher bereitstellen
- Betriebssystem gibt jedem Prozess einen virtuellen Speicher isoliert von anderen Prozessen
  - ⇨ Jeder Prozess Adressrum von 0 – max => homogen & größer als verfügbarer RAM
  - ⇨ Vereinfachung der Programmierung

Vorteile:

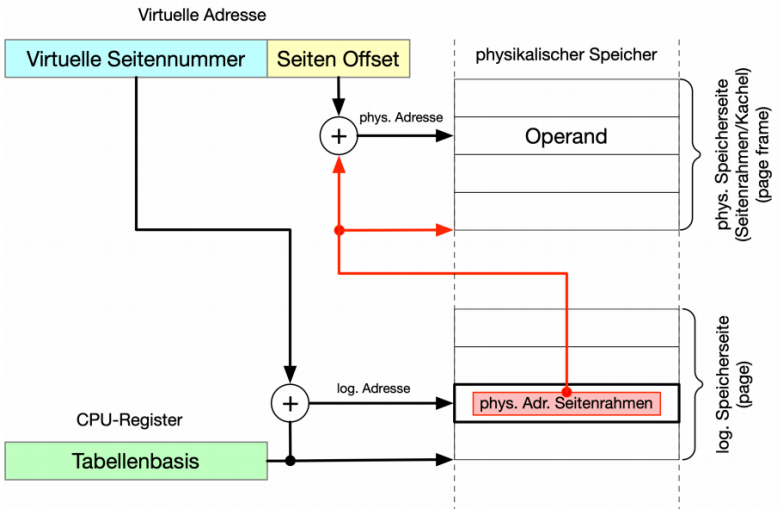
- Der virtuelle Speicher ermöglicht jedem Prozess die Nutzung eines eigenen Adressraums, der
  - bedeutend größer ist als der im Computer verfügbare RAM
  - und der jeden Prozess einheitlich ab Adresse 0 beginnt.
- Die virtuellen Speicher verschiedener Prozesse sind voneinander isoliert.

Nachteile:

- Zur schnellen Abbildung von virtuellen Adressen auf physische Adressen wird eine Hardwareunterstützung benötigt (MMU)
  - Das Konzept des virtuellen Speichers erfordert eine Abbildung von virtuellen Adressen auf physische Adressen
  - Die Abbildung der virtuellen auf physischen Adressen findet in der MMU statt.

Grundprinzip der virtuellen Adressierung

- Segment: 1 Bereich aus Prozesspeichermodell
- Seite: 1 Bereich aus logische (virtuelle) Adressbereich
- Seitenrahmen: 1 Bereich aus physikalischen Adressbereich (RAM)
- Block: kleinste Dateneinheit auf Festplatte – 512 Byte



Swapping: Das Auslagern von kompletten Prozessen (Prozess-Adressräumen) auf den Hintergrundspeicher (Festplatte) wird Swapping (engl. tauschen) genannt.

Paging: Das Auslagern von Teilen von Prozessen (Segmente) wird Paging (engl. Seitenverwaltung) genannt.

4 LZ_04_Betriebssysteme

4.1.1 Schichten moderner Betriebsysteme

Die Schichtenarchitektur von modernen Betriebssystemen lässt sich, angefangen bei den Funktionen der Hardwareunterstützung aufsteigend, folgendemaßen beschreiben.

1. Softwareschicht zwischen der Hardware und dem Kern des Betriebssystems (Kernel).
2. Elementare Funktionen des Kernels.
3. Ein.- Ausgabeverwaltung, Netzwerkdienste.
4. Speicher und Dateiverwaltung.
5. Fehlerbehandlung.
6. Bedien.- und Programmierschnittstelle. (vgl. Die Grundaufgaben des Betriebssystems.)

4.1.2 Dateisysteme

Eine Datei (engl. file) ist eine logische, inhaltlich zusammengehörenden, Einheit von Daten

4.1.3 Praktische Dateiverwaltung

Kommandos zum Umgang mit Dateien und Verzeichnissen

Kommando	Bemerkungen
`pwd`	Aktuelles Verzeichnis ausgeben
`cd`	Aktuelles Verzeichnis wechseln
`ls`	Inhalt von Verzeichnissen auflisten
`cat <Datei>`	Dateiinhalt ausgeben
`more <Datei>`	Dateiinhalt seitenweise ausgeben
`less <Datei>`	Dateiinhalt seitenweise ausgeben (Blättern u. Suchen möglich)
`cp <alte Datei> <neue Datei/Verzeichnis>`	Datei kopieren
`mv <alte Datei> <neue Datei/Verzeichnis>`	Umbenennen/verschieben
`rm <Datei>`	Datei löschen
`rm -r <Verzeichnis>`	Verzeichnis rekursiv löschen
`mkdir <Verzeichnis>`	Verzeichnis anlegen
`rmdir <Verzeichnis>`	Verzeichnis löschen (muss leer sein)

Echo -> modifiziert

<u>Typen von Verweisen:</u>

1. Feste Verweise, Hard-Links (engl. hard link):

Hierbei wird im zweiten Verzeichnis einfach eine Datei mit derselben I-Node Nummer wie die der originalen Datei eingetragen. Hard-Links sind nur innerhalb einer Partition erlaubt (möglich)! Verzeichnisse sind nicht erlaubt.

Ln Datei Dateihardlink

2. Symbolische Verweise, Symbolische-Links (engl. soft link):

Hierbei wird im zweiten Verzeichnis eine spezielle Datei angelegt, in der nur der Pfadname zur „Originaldatei“ im ersten Verzeichnis steht. Symbolische-Links können überall in den Verzeichnisbaum zeigen.

Ln – s Datei neueDateiname

<u>hmod-Kommando</u>: können die Bits für die Zugriffrechte geändert werden.

- Besitzer **user u** darf lesen (r) /schreiben (w) /ausführen (x)
- Gruppe **users g** darf lesen (r) / ausführen (x) (aber nicht schreiben (w))
- Andere **other o** dürfen lesen (r) (aber nicht schreiben (w) oder ausführen (x))

chmod **u**-x Dateiname

4.2 Praktische Prozessverwaltung NKR

4.2.1 Informationen über Prozesse:

Das ps-Kommando erstellt eine Liste der Prozesse und zeigt Informationen zu den Prozessen an.

Zu jedem Prozess wird eine Zeile ausgegeben, in der u. a. folgende Informationen stehen:

- Die Prozessnummer (PID, Process Identifier).
- Die Prozessnummer (PPID, Parent Process Identifier) des Elternprozesses. Das
- Terminal (TTY, TeleTYpe) aus dem heraus der Prozess gestartet wurde.
- Das Kommando (CMD, COMMAND), durch das der Prozess gestartet wurde.

Auswahl von Optionen des ps-Kommandos:

- ps -u: Zeigt die Besitzer der Prozesse (user).
- ps -e: Zeigt alle Prozesse, nicht nur die Eigenen.
- ps -f: Ausführliche Informationen ausgeben (full).
- ps -l: Vollständige Informationen ausgeben (long).
- ps -ef --forest: Prozessliste mit Baumdarstellung.
- ps -fo user,ppid,pid,tt,fname: Benutzerdefinierte Ausgabe.

4.2.2 Verwaltung von Prozessen

- Jeder Prozess kann einen neuen Prozess erzeugen.
- In einer Shell kann ein Programm auch im Hintergrund gestartet und ausgeführt werden.
- Prozesse können miteinander kommunizieren, indem sie sich gegenseitig Signale zusenden.

Aktuelle Prozessübersicht:

Das jobs-Kommando zeigt alle PID‘s und Job-Nummern von Prozessen an, die mit der aktuellen Shell verbunden sind.

Aktivieren in Vorder.- oder Hintergrund:

- Mit dem **fg-Kommando** können suspendierte Prozesse wieder aktiviert und in den Vordergrund gestellt werden: fg %.
- Mit dem **bg-Kommando** können suspendierte Prozesse wieder aktiviert und in den Hintergrund gestellt werden: bg %.

Prozessname oder Prozessnummer:

- Mit dem killall-Kommando werden Prozesse mit Hilfe ihres Namens beendet.
  Beispiel: killall -e xosview
- Ist nur der Prozessname bekannt, so kann mit den Kommandos „pgrep“ oder „pidof“ die Prozessnummer PID ermittelt werden.
  Beispiel: pgrep xosview.

Prozesskommunikation mit Signalen

Kommunikation mit Prozessen über eine Shell:

- **Prozesse** können **miteinander kommunizieren**, indem sie sich gegenseitig **Signale zusenden**.
- Eine **Shell** ermöglicht es einem **Benutzer**, **Signale** an einen **Prozess** zu **senden**.

Senden von Signalen mit der Tastatur:

Signal	Tastenkombination	Bedeutung
`suspend`	`CTRL + z`	Hält den Vordergrund-Prozess an.
`suspend`	`ENTER, ~, CTRL + z`	Hält den Vordergrund-Prozess an, wenn eine SSH-Verbindung genutzt wird.
`kill`	`CTRL + c`	Abbruch des Prozesses.
`exit`	`CTRL + d`	Beendet den Prozess der Login-Shell.

Senden von Signalen mit dem `kill`-Kommando:

- Mit dem `kill`-**Kommando** können **Signale** an **Prozesse gesendet** werden.
  Beispiel: `kill -SIGSTOP <PID>`, `kill -SIGCONT <PID>`, `kill -9 <PID>`.
- **Signalliste** mit `kill -ℓ`. Weitere Infos unter `man -7 signal` oder `man -k signal`

4.3 Programmieren mit der Bash

4.3.1 Ausführen eines Bash-Skripts

Dateizugriffsrechte werden mit dem chmod-Kommando angepasst.

user@jupiter :~ $ chmod u + x ./ halloWelt . sh

```
1 #!/ bin / bash
2 # Ich bin ein Kommentar 3
3 echo " Hallo Welt !";
```

Echo = print

- **Beispiel** für ein **Bash-Skript** mit **Parameterübergabe** (*parameterTest.sh*):

```
#!/bin/bash
echo "Mein Name ist $0";
echo "Anzahl der Parameter = $#";
echo "Parameter 1 = $1";
echo "Parameter 2 = $2";
```

- Beim **Aufruf** eines **Bash-Skripts** werden die **Parameter** (*hier* `foo`, `bar`) durch **Leerzeichen** voneinander **getrennt** übergeben. **Beispiel**:

```
user@jupiter:~$ ./parameterTest.sh foo bar
Mein Name ist ./parameterTest.sh
Anzahl der Parameter = 2
Parameter 1 = foo
Parameter 2 = bar
```

4.3.2 Deklaration | Bash-Variablen

```
1 varA=Hallo ; 2
2 varB=" Hallo Welt !";
```

Kein Leerzeichen! Die Variablen der Bash besitzen keinen Typ (int, float, char, etc.).

Zugriff auf den Wert einer Variablen

Auf den **Wert** (den Inhalt) einer **Variablen** wird mit Hilfe des $-**Zeichens zugegriffen**.

```
#!/bin/bash
myPath="/home/";
myUID="maxi";
echo "$myPath$myUID"; #Schlecht lesbar
echo "${myPath}${myUID}"; #Besser lesbar
```

- **Zeile 2**: **Deklaration** der **Variablen** „`myPath`“ mit dem **Wert** „`/home/`“.
- **Zeile 3**: **Deklaration** der **Variablen** „`myUID`“ mit dem **Wert** „`maxi`“.
- **Zeile 4**: **Ausgabe** der **Variablen** „`myPath`“ und „`myUID`“.
- **Zeile 5**: Wie Zeile 4. **Variablennamen** sind mit den **Klammerpaaren** {} abgegrenzt (*besser lesbar, guter Programmierstil*).

Wird eine **nicht deklarierte Variable** zur **Ausgabe** genutzt, so wird eine **leere Zeichenkette** der **Länge 1 ausgegeben**.

Variablenname	Bedeutung
$#	Anzahl der übergebenen Argumente.
$0	Name des ausgeführten Skripts.
$1	Wert des 1. übergebenen Arguments.
$2 ...	Wert des 2. übergebenen Arguments.
${10} ...	Wenn mehr als 9 Argumente übergebenen werden.

Einfache Hochkommas 'a' (Single-Quote) =
Variablen nicht durch ihre Werte ersetzt werden.
Doppelte Hochkommas "a" (Double-Quote) =
Variablen durch ihre Werte ersetzt werden.
Back-Quotes `a` (Kommando-Substitution) =
der als Kommando interpretiert. (Rückgabewert = var)
Die Bash- und die Korn-Shell unterstützen die erweiterte Syntax „$(. . .)“ für Back-Quotes.

Verkettung von Kommandos

Der **Pipe-Operator** „|“ leitet die **Ausgabe** eines **Kommandos** als **Eingabe** an ein **anderes Kommando weiter**.

- **Beispiele**:

```
ls -1d /etc/*/ | more;
cat /var/log/auth.log | tail -n 3
cat /etc/passwd | grep "root" | cut -d':' -f7
```

- **Zeile 1**: **Ausgabe** aller **Unterverzeichnisse** von `/etc` und **seitenweises Anzeigen** mit `more`.
- **Zeile 2**: **Ausgabe** der **Datei** „`/var/log/auth.log`“ und **Anzeige** der **letzten 3 Zeilen**.
- **Zeile 3**: **Ausgabe** der **Datei** „`/etc/passwd`“, **filtern** nach **Zeilen** die den **Benutzernamen** „`root`“ **enthalten** und **Ausgabe** des **7**-ten **Feldes** wenn das **Zeichen** „`:`“ als **Trennzeichen betrachtet** wird.

Umlenken der Aus-/Eingabe in eine Datei

Mit den **Operatoren** „>“ und „>>“ kann die Ausgabe eines **Kommandos** in eine **Datei umgeleitet** (*geschrieben*) werden.
Mit dem **Operator** „<“ kann die Eingabe für ein **Kommando** aus einer **Datei gelesen** werden.

```
ls -1d /etc/*/ > ./folders.txt;
ls -1d /etc/*/ >> ./folders.txt;
grep "root" < /etc/passwd;
```

- **Zeile 1**: Die **Ausgabe** des **Kommandos** „`ls`“ wird in die **Datei** „`./folders.txt`“ **geschrieben**. Falls die Datei „`./folders.txt`“ nicht existiert wird sie erstellt. Vorsicht!. Die Datei „`./folders.txt`“ wird **überschrieben** und ihr **bisheriger** Inhalt geht **verloren**!
- **Zeile 2**: Die **Ausgabe** des Kommandos „`ls`“ wird an das Ende der **Datei** „`./folders.txt`“ **angehängt**.
- **Zeile 3**: Die Datei „`/etc/passwd`“ wird eingelesen und als **Eingabe** für das **Programm** „`grep`“ verwendet (Eingabeumleitung).

Beispiel: Vergleich auf „ var1 größer als var2“.

```
#!/bin/bash
num1=50;
num2=127;
if [ ${num1} -gt ${num2} ]; then
 echo "${num1} ist groesser als ${num2}";
else
 echo "${num1} ist kleiner oder gleich ${num2}";
fi
```

Ausdruck	Operator	Ist wahr (0), wenn...
["${var1}" = "${var2}"]	=	var1 gleich var2 ist
["${var1}" != "${var2}"]	!=	var1 ungleich var2 ist
[-z "${var1}"]	-z	var1 leer ist
[-n "${var1}"]	-n	var1 nicht leer ist

Übersicht der Vergleichsoperatoren:

Ausdruck	Operator	Ist wahr (0), wenn...
[${var1} -eq ${var2}]	eq = equal	var1 gleich var2 ist
[${var1} -ne ${var2}]	ne = not equal	var1 ungleich var2 ist
[${var1} -lt ${var2}]	lt = less then	var1 kleiner als var2 ist
[${var1} -gt ${var2}]	gt = greater then	var1 größer als var2 ist
[${var1} -le ${var2}]	le = less equal	var1 kleiner oder gleich var2 ist
[${var1} -ge ${var2}]	ge = greater equal	var1 größer oder gleich var2 ist

Operator/Ausdruck	Bedeutung
[-e <DATEI>]	<DATEI> existiert
[-b <DATEI>]	<DATEI> existiert und ist ein **Block Special Device** (Gerätedatei)
[-c <DATEI>]	<DATEI> existiert und ist ein **Character Special File** (Gerätedatei)
[-d <DATEI>]	<DATEI> existiert und ist ein **Verzeichnis**
[-f <DATEI>]	<DATEI> existiert und ist eine **reguläre Datei**
[-h <DATEI>]	<DATEI> existiert und ist **symbolischer Link** (oder -L)
[-p <DATEI>]	<DATEI> existiert und ist eine **Named Pipe**
[-S <DATEI>]	<DATEI> existiert und ist ein **UNIX Domain Socket**

Die `for`-Schleife

Die `for`-**Schleife** der **Bash** führt eine **Befehlsfolge** für eine **angegebene Liste** von **Wörtern** (`<wort1>`, `<wort2>`, ...) aus.

Allgemeine Syntax:

```
for var in <wort1> <wort2> <wort3>; do
 <1-te Kommando>;
 ...
 <n-te Kommando>;
done
```

1. Vor jedem **Eintritt** in die **Schleife** wird das **nächste Wort** der **Liste** (Schlüsselwort `in`) in die **Variable** „`var`“ **übertragen**.
2. Die **Kommandos** zwischen den **Schlüsselworten** „`do`“ und „`done`“ werden **ausgeführt**. Danach **beginnt** der **nächste Schleifendurchlauf**.
3. Dieser **Vorgang** wird so lange **wiederholt**, bis **alle Elemente** (Wörter) der **Liste abgearbeitet** sind.
4. **Fehlt** das **Schlüsselwort** `in` und die **Wörterliste**, dann werden automatisch die dem **Skript** übergebenen **Argumente** als **Wortliste verwendet**.

Die `while`-Schleife

Bei der `while`-**Schleife** werden die **Kommandos zwischen** den **Schlüsselwörtern** „`do`“ und „`done`“ solange **abgearbeitet** wie die `<Bedingung>` **wahr** ist.

Allgemeine Syntax:

```
while [ <Bedingung> ]; do
 1-te Kommando;
 ...
 n-te Kommando;
done
```

- Trifft die `<Bedingung>` **nicht mehr zu** und ist **falsch**, wird die **Schleife beendet** und die **Ausführung** des **Scripts hinter** dem Schlüsselwort „`done`“ **fortgeführt**.

4.3.3 Programm das wahrscheinlich in der Klausur drankommen wird!!!

Gegeben sei eine Textdatei mit Adresseinträgen. In jeder Zeile der Datei steht ein Adresseintrag mit den folgenden Adressinformationen:

• Vorname Name, Strasse, Hausnummer, Postleitzahl, Ort

Mit dem Shellskript suchen wir Leute mit Kriterien raus.

```
#!/bin/bash

# Terminal löschen
clear;

#Anzahl der Parameter ok?
if [ $# -lt 2 ]; then
       echo "usage: $(basename $0) <Dateiname> <Suchbegriff>";
       echo "$(basename $0): Programmausfuehrung beenden";
       exit 0;
fi

# Existiert die angegebene Datei?
if [ ! -f "./$1" ]; then
       echo "$(basename $0): Die Datei \"$1\" existiert nicht!";
       echo "$(basename $0): Programmausfuehrung abgebrochen!";
       exit 0;
fi

# Durchsuchen der Adressdatei $1 nach 1. Suchmuster $2
searchResult=$(fgrep "$2" "$1");

# Verschiebt Argumentenliste nach rechts, sodass $1 verschwindet
shift;

# Durchsuche alle Suchmuster durch Argumentenliste
for searchItem; do
    searchResult=$(echo "${searchResult}" | fgrep "${searchItem}";
done

echo "${searchResult}";

echo;
echo "$(basename $0): Programmausfuehrung beenden";
```

- *.jpeg bedeutet alle jpeg Dateien in diesem Verzeichnis

```
#!/bin/bash

for fileName in *.jpeg; do
       fileBaseName=$(basename ${fileName} .jpeg);
       mv ${fileBaseName}.jpeg ${fileBaseName}.jpg;
done

exit 0;
```

5 LZ_05_SE-Werkzeuge

Aufgabe 4: Wiederholen Sie die Funktionsweise der Adressierungsmodi des Intel-Assemblers. Schreiben Sie sich fur jeden der folgenden Modi auf, wie man ihn benutzt und wie er funktio- ¨ niert.
a) Register Mode Adressierung b) Immediate Mode Adressierung c) Memory Mode Adressierung

6 LZ_06_Informationssysteme

6.1 Datenbanken

System zur Beschreibung, Speicherung und zum Abrufen von großen Datenmengen

6.2 Logische Modelle - Entity Relationship Modell, kurz ER-Modell

- Basiert auf den Grundkonzepten Entity (Informationseinheit), Attribut (Eigenschaft eines Entity) und Relationship (Beziehung zwischen Entities)

6.3 Das relationale Modell

Der Relationenname (R) und die Attributnamen (A1, A2, A3) zusammen mit der Domäne (Menge der möglichen atomaren Attributwerte) bilden das Relationenschema

Tupel = Ein Datensatz (Bsp. eine Zeile ist ein **Tupel** in einer Tabelle (ein Datensatz))

6.4 SELECT Anweisungen KR!!!!

- *SELECT* = Spalte die man ausgegeben haben möchte
- *FROM* = Auswählen von der Tabelle
- *WHERE* = Eine Spalte, wo man Bedingungen eingibt (and, or, not (Zusatzinfos))

Geben Sie alle Namen und Matrikelnummern von Studenten aus.

```
SELECT name, Martikelnummer
FROM R1
```

Geben Sie alle Matrikelnummern von Studenten aus, die im Modul GDI einen zweiten Versuch unternommen haben.

```
SELECT Martikelnummer
FROM R2
WHERE Versuche = „2“ AND Modulu = „GDI“
```

Schreiben Sie eine SQL-Anweisung, die das kartesische Produkt von R1 und R2 ausgibt.

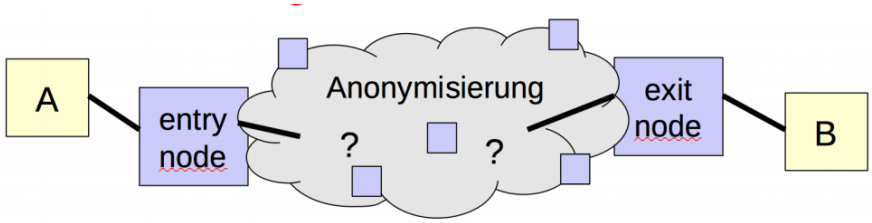
```
SELECT *
FROM R1, R2
```

- Geben Sie eine SQL-Anweisung an, die die Namen aller Studenten ausgibt, die im Fach GDI eine 1,3 geschat haben.

```
SELECT name
FROM R1,R2
WHERE Ergebnis = „1,3“ AND (Martikelnummer = Student) AND Modulu =
„GDI“
```

6.5 Anonyme Internetnutzung mit TOR (genannt Onion Router)

- TOR-Prinzip: Kombiniere asymmetrische Kryptographie mit vielen Zwischenstationen
- Absender A verschlüsselt zuerst für B, dann noch mal für den exit node, dann nochmal für OR vor dem exit node, . . . dann für den entry node.
- Diese x-mal verschlüsselten Daten werden dann zum entry node geschickt.
- Entry-Node entschlüsselt und weiß dann welches der nächste OR ist. Jeder OR entschlüsselt und leitet weiter.
- Jeder Router kennt nur den unmittelbaren Vorgänger und den unmittelbaren Nachfolger.



7 LZ_07_Netze

7.1 Klassifikationen NKR

Mehrere miteinander verbundene autonome Computer

7.1.1 Übertragungstechnik

Punkt-zu-Punkt-Verbindungen:

- Dedizierte Leitungen zwischen Rechnern
- Vermittlungsstationen

Broadcast-Netze:

- Ein einziger Übertragungskanal
- Protokolle für Zugriffskontrolle

7.1.2 Logische Bezüge

Peer-to-Peer

- Gleichberechtigung der beteiligten Rechner
- keine Sonderaufgaben für einzelne Rechner

Client-Server

- Clients (Rechner) wollen bestimmten Dienste benutzen
- Server bieten diese Dienste an

7.1.3 Übertragungsreichweite

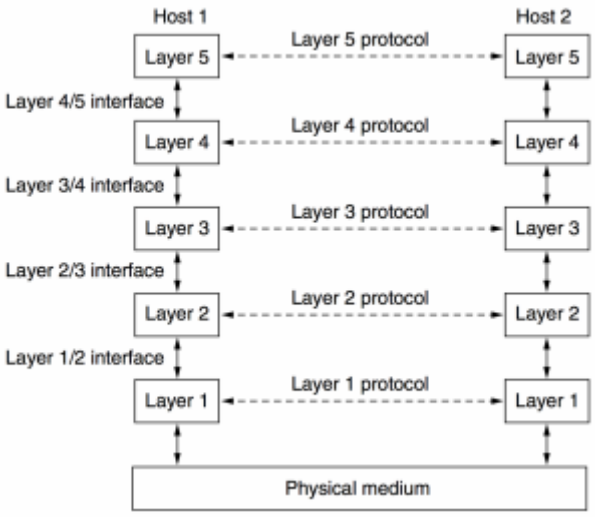
- LAN - Local Area Network (meist Broadcast-Netzwerke): 1km – 10km
- MAN - Metropolitan Area Network (ähnlich wie bei LAN): > 10km
- WAN - Wide Area Network (meist Punkt-zu-Punkt-Verbindungen): > 1000km
- Verbundnetze: Vernetzung von WANs

7.1.4 Schichtenarchitektur

- Problem in mehrere Teile zerlegen => Schichten
- Netzsoftware aus mehreren übereinander gelagerten Schichten
- Jede Schicht stellt der nächsthöheren Schicht Dienste zur Verfügung => **Schnittstelle**
- Alle Schichten **implementiert Protokolle** => **Zusammenfassung** aller **Protokoll-stack** genannt

Aufbau

- Beide Hosts folgen den gleichen Protokollen in den jeweiligen Schichten, um miteinander zu kommunizieren



Ablauf der Kommunikation

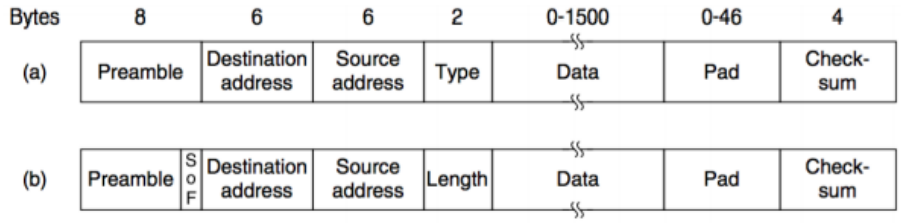
- Jede Schicht fasst Daten mit dem Protokoll zusammen und verwendet die Dienste der darunterliegenden Schichten
- Daten zum Host 2 geschickt, werden alle Schichten Daten von den darunterliegenden Schichten zugreifen, interpretieren diese Daten mit deren Protokoll und liefern diese Daten der nächsthöheren Schicht weiter

7.2 Lokale Netze, Ethernet und WLAN

7.2.1 Ethernet LAN

- Es werden Datenrahmen übertragen
- Es gibt das klassische Ethernet und switched Ethernet
- Switch stellt die Pakete von einem Eingangsport den richtigen Ausgangsport zu
  => erstellt eine Tabelle von Adressen und Ports

Rahmenformat



- Präambel: dient der Synchronisation zwischen Empfänger
- Padding: zu kurze Rahmen werden auf die mindestlänge 64 Byte gefüllt

7.2.2 Ethernet-Adressen / MAC-Adresse Media Access Control

- Hexadezimal => 6 Byte
- Einzeladressen beginnen mit 0
- Gruppenadressen beginnen mit 1
- Multicast-Adressen: spricht eine Gruppe von Rechnern an
- Broadcast-Adresse: FF:FF:FF:FF:FF:FF => spricht alle Rechner im Netzwerk an

⇨ In lokalen Netzen sollten Ethernet-Adressen eindeutig sein

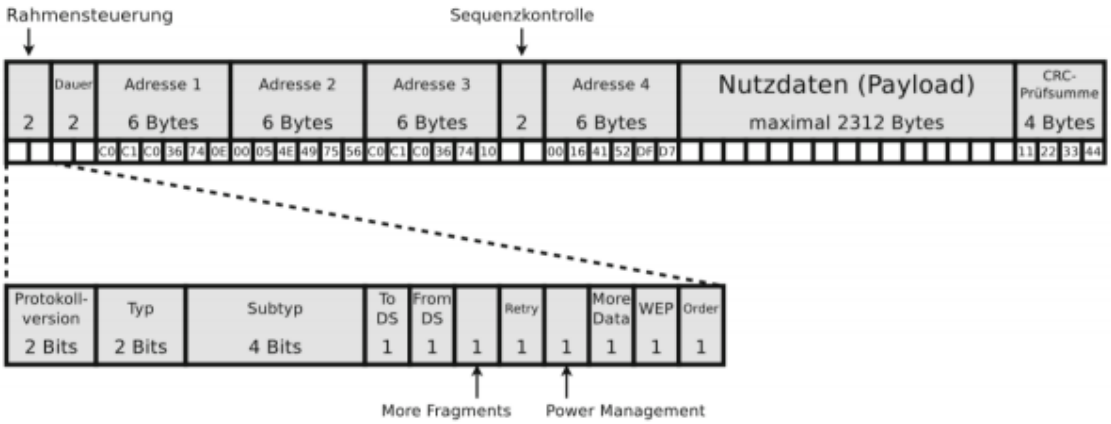
7.2.3 WLAN-Betriebsmodi

Ad-hoc Modus 2 WLAN-Stationen kommunizieren direkt miteinander

Infrastruktur-Modus 2 WLAN-Stationen kommunizieren nicht direkt miteinander, sondern über einen Access Point

7.2.4 WLAN-Rahmenformat

- Mehrere Adressen, da über Access Point die Pakete vermittelt werden und die Access Points müssen auch adressiert werden



7.2.5 Wichtigsten Eigenschaften

- **Komplexere Zugriffskontrollen** auf das Übertragungsmedium und eine Kollisionserkennung sind erforderlich

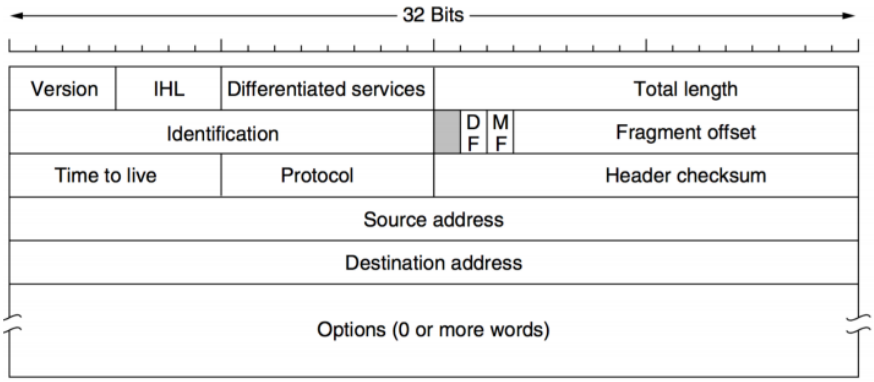
  ⇨ Niedrigere Datenübertragungsraten als bei kabelgebundenen Netzen
- Sicherheitsprobleme

7.3 Vernetzte Netzte – Internet KR!!!

7.3.1 IPv4-Adressen

- 4 Byte (32 Bit lang) z.B.: 141.71.31.220 => 2^32 Adressen
- Adressen bestehen aus Netz- und Rechneradresse
- Subnetzmaske gibt an wo **Netz**- und **Rechneradresse** ist, indem sie angibt wie viele Bits Netzadresse hat. Schreiweisen:
  - ⇨ 130.97.16.132/26 => 26 Bits hat Netzadresse
  - ⇨ 130.97.16.132/255.255.255.192 => oder als Bitmaske

7.3.2 Aufbau von IPv4-Paketen



Version: Protokollversion (z.B.: 4)

- IHL = Länge des Headers
- Type of Service: gewünschter Service
- Total Length: Länge Header & Nutzdaten
- Identification: Zugehörigkeit von Fragmenten zu Datengrammen
- DF: don‘t fragment
- MF: more fragments
- Fragment Offset: gibt die Stelle im Datengramm an, an die das Fragment gehört
- Time to live (TTL): Lebenszeit des Paketes
- Protocol: gibt an, zu welchem Protokoll die Nutzdaten gehören
- Header Checksum: Prüfsumme zur Erkennung von Fehlern im Header

7.3.3 Paketversand innerhalb eines lokalen Netzes

- Netzadresse von Absender und Zielrechner müssen gleich sein
- IP-Adresse bekannt und MAC Adresse bekannt:
  - ➢ Absender schickt Ethernet Paket in dem sich ein IP Paket sich befindet an den Zielrechner und interpretiert das Ethernet Paket
- IP-Adresse bekannt und MAC Adresse unbekannt - Address Resolution Protocol (ARP):
  - ➢ Absender schickt ein ARP request, welcher mit der Broadcast Adresse an alle Rechner im lokalen Netzwerk geschickt wird. Dabei handelt es sich um ein Ethernet Paket in der die IP-Adresse vom Absender und des Zielrechners sich befindet. Alle Rechner ignorieren den ARP request bis auf der Zielrechner, dieser antwortet mit dem ARP reply seine MAC Adresse an den ursprünglichen Absender. Die IPv4-Adressen werden auf MAC-Adressen abgebildet in der ARP-Tabelle für die Zukunft

7.3.4 Paketversand an Rechner in einem anderen Netz

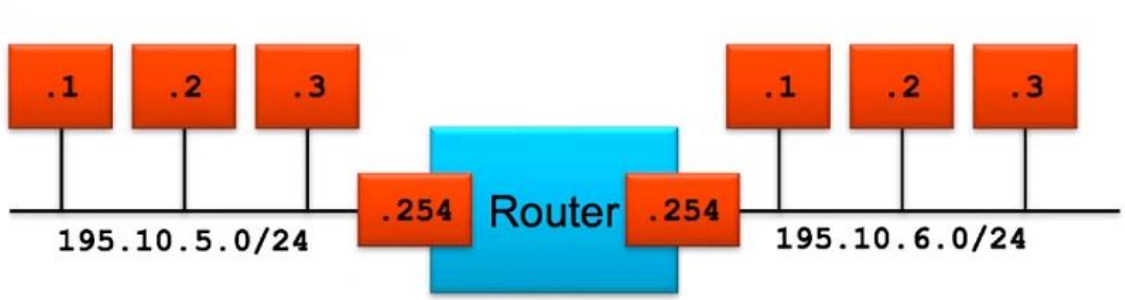
- Netzadresse von Absender und Zielrechner unterschiedlich
- Ethernet Paket mit MAC-Adresse vom Standardgateway (ARP falls nicht bekannt) in dem sich das IP-Paket des Zielrechners befindet ans Standardgateway schicken, der Router
- Routing über mehrere Router

7.4 Wichtige Internetdienste

7.4.1 Domain Name Service (DNS)

- Namen für IPv4 Adressen => besser zum Merken
- werden durch Punkte getrennt z.B.: vmhost.inform.hs-hannover.de
- DNS-Server können zu den Namen die IPv4-Adressen bestimmen

7.5 Klausur Aufgabe



Wie funktioniert der Versand von 195.10.5.1 nach 195.10.6.3 eines IP-Pakets?

Zum Versand eines IP-Pakets benötigt man die MAC-Adresse.
Als erstes wird die Subnetzadresse von beiden Rechnern gebildet und verglichen.
Die Ziel-IP-Adressen befinden sich nicht im lokalen Netzwerk.
=> ein Ethernetframe, in der sich die IP-Adresse dies Zielrechners befindet, wird an den Router geschickt.
Der Router benutzt das ARP-Protokoll:
Durch einen ARP-request wird ein MAC-broadcast erstellt. Das bedeutet das ein Paket an alle Rechner geschickt mit der Ziel-IP-Adresse und falls die gefunden wird antwortet der Rechner mit einer ARP-reply in der sich die MAC Adresse des Ziel-Rechners befindet. Und der Router schickt die MAC-Adresse an den Source-Rechner. Und die IP-Adresse kann auf die MAC-Adresse abgebildet werden.
Nun ist ein Paketversand vom Source-Rechner an den Ziel-Rechner möglich.

8 LZ_08_Internet

8.1 Telnet

8.1.1 Telnet Einsatzgebiete

- Analyse von textbasierten Internet-Protokollen

8.1.2 Eigenschaften von Telnet

- Daten unverschlüsselt, deshalb durch SSH ersetzt
  - ⇨ Sicherheitsbedenken

8.1.3 Benutzung von Telnet

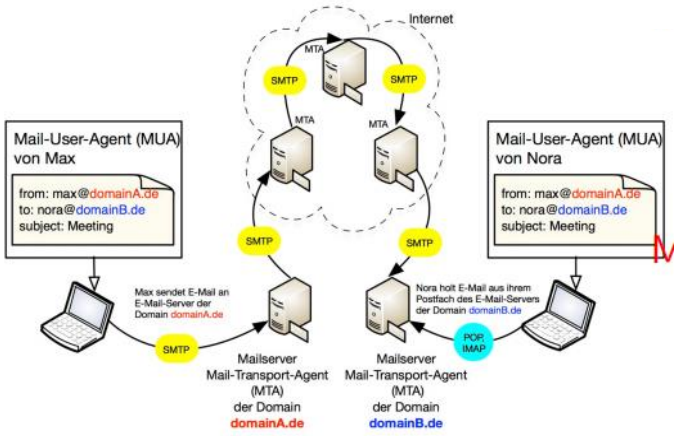
- „telnet“ [IP-Addr / DNS-Name] [Port-Nr.]
- telnet ohne Ziel führt in den Befehlsmodus

Befehl	Bemerkung
Close	Schließt die aktuelle Telnet-Session.
Display	Zeigt aktuelle Telnet-Parameter-Einstellung
Open	Baut die Verbindung zu einem Telnet-Server-Prozess auf
Quit	Baut aktuelle Telnet-Verbindung ab und deaktiviert den Telnet-Prozess im lokalen Rechner
Set	Ermöglicht das Setzten von Telnet-Parametern (z.B. echo, CRLF, Escape, etc.)
Status	Zeigt Status und gesetzte Parameter der aktuellen Session
Unset	Hebt Parameter-Festlegungen auf

8.2 Simple Mail Transfer Protocol ()

8.2.1 Übersicht über E-Mail-Versand im Internet

- **Mail User Agents (MUA):**
  - ⇨ Erstellen und versenden E-Mails
  - ⇨ verwalten lokale Postfächer
- **Mail Transport Agents (MTA):**
  - ⇨ Entgegennehmen, Zwischenspeichern, Weiterleiten von E-Mails
  - ⇨ Verwalten Postfächer der Benutzer auf dem Server



8.2.2 Überblick Simple Mail Transfer Protocol (SMTP)

- Textbasiertes Protokoll
- Art des Transports irrelevant
- Anforderung des Transports:
  - ⇨ Zuverlässiger Datenstrom
  - ⇨ Geordneter Datenstrom (Reihenfolge bleibt erhalten)
  - ⇨ Kommunikation in beide Richtungen

8.2.3 Ablauf bei E-Mail-Versand mit SMTP

1. Verbindung zum Empfänger-SMTP-Server (Port-Nummer 25) aufbauen
2. Der Absender-SMTP-Server „identifiziert“ sich gegenüber dem Empfänger-SMTP-Server
3. Absender-SMTP-Server führt eine oder mehrere E-Mail-Übertragungen (Transaktion) aus bestehen aus Teilschritte
    1. Übertragen Absender-Information
    2. Übertragen Empfänger-Information
    3. Übertragen der Nachricht
4. Absender-SMTP-Server beendet Verbindung

8.2.4 Allgemeine Anmerkungen zu SMTP

- Nur einzelne (Text-)Zeilen werden gesendet, Client Kommandos & Server Status

Client Kommandos		Server Status	
HELO	Begrüßungskommando	220	Gruß des SMTP-Servers
MAIL FROM:	Absenderangabe	250	Befehl erfolgreich
RCPT TO:	Empfängerangabe	354	Beginne Nachricht
DATA	Beginne Nachricht	221	Schließe Verbindung

8.2.5 SMTP Beispielübertragung

```
telnet smtp.hs-hannover.de 25 # Verbindungsaufbau (TCP) zu Port 25 des SMTP-Servers

HELO hs-hannover.de # Identifikation des Absenders

MAIL FROM: <detijon.lushaj@stud.hs-hannover.de> # Durchführen der Mail-Transaktion
RCPT TO:<stefan.wohlfeil@hs-hannover.de>
DATA # E-Mail-Text Eingabe
From: Detijon Lushaj (detijon.lushaj@stud.hs-hannover.de)
To: Stefan Wohlfeil (stefan.wohlfeil@hs-hannover.de)
Subject: SPAM von Detijon Lushaj          #FROM, TO & SUBJECT reicht als Header, der Rest ist optional
                                          #Leerzeichen wichtig!
HIER KOMMT MEIN TEXT LOL
LOL

. #Mit „.“ Nachricht zu Ende
QUIT #Beenden
```

8.2.6 Eigenschaften von SMTP

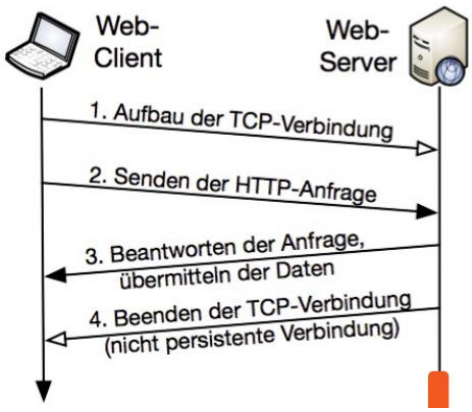
- E-Mail-Übertragung Unterscheidung zwischen Nachricht (message) und Umschlag (envelope)
- Message:
    - ⇨ alles in DATA
    - ⇨ besteht aus Kopf (head) und Rumpf (body)
    - ⇨ Absender (FROM) und Empfänger (TO) wiederholt, diese werden von MUA angezeigt
    - ⇨ Nur ASCI Zeichen erlaubt
- **Envelope**: enthält Absender (MAIL FROM) und Empfängern (RCPT TO). An diese Adressen wird die E-Mail zugestellt

Kopf-Zeile	Beschreibung
Subject:	Betreff der E-Mail
From:	Absender der E-Mail
To:	Empfänger der E-Mail
Cc:	Weitere Empfänger der E-Mail
Bcc:	Weitere Empfänger der E-Mail (ohne, dass die anderen Empfänger davon wissen)
Organization:	Zu welcher Organisation gehört der Absender
Reply-To:	An welche Adresse sollen Antworten geschickt werden

8.3 Hypertext Transfer Protocol (HTTP)

8.3.1 HTTP Übersicht

- zustandsloses Protokoll => Informationen zu früheren Sitzungen gehen verloren, kann durch Cookies aufgehoben werden
- Übertragung von Daten im Klartext zwischen WWW-Client und WWW-Server
  - ⇨ WWW-Client: Web-Browser
  - ⇨ WWW-Server: Web-Server
- **Verbindungsaufbau (TCP) zu Port 80**



8.3.2 HTTP 1.0: Protokoll-Ablauf-Schema

1. Web-Browser baut TCP-Verbindung zum Web-Server auf
2. Nach Verbindungsaufbau sendet Web-Browser die HTTP-Anfrage (http request) an Web-Server
3. Web-Server antwortet mit HTTP-Antwort (http response) mit angefragten Daten
4. Web-Server beendet TCP-Verbindung zum Web-Browser

8.3.3 HTTP: Kommunikation

- Jede Nachricht (Anfrage und Antwort) besteht aus zwei Teilen
  1. Nachrichtenkopf (message head) enthält Metadaten
  2. Nachrichtenrumpf (engl. message body) enthält Nutzdaten wie HTML-Seite

HTTP-Methode	Beschreibung
GET	Anfordern einer Ressource (z. B. Datei) unter Angabe des Ressourcennamens oder einer URL von einem Web-Server
HEAD	Wie GET. Allerdings wird nur der Antwortkopf vom Web-Server gesendet
PUT	Speichert eine Ressource unter Angabe einer Ziel-URL auf einen Web-Server (hochladen).
DELETE	Löscht eine Ressource unter Angabe einer Ziel-URL auf einem Web-Server
OPTIONS	Liefert eine Liste der vom Web-Server unterstützten Methoden bzw. Merkmalen

Status-Code-Gruppe	Beschreibung
1xx: Informational	Die Anfrage wurde empfangen und wird weiter bearbeitet.
2xx: Success	Die Anfrage wurde ordnungsgemäß empfangen, bearbeitet und eine Antwort wird an den Anfragesteller zurückgesendet.
3xx: Redirection	Die Anfrage war gültig, jedoch ist die Ausführung von weiteren Schritte seitens des Clients erforderlich.
4xx: Client Error	Die Anfrage wurde aufgrund einer ungültigen Syntax nicht ausgeführt. Verantwortungsbereich des Web-Clients.
5xx: Server Error	Der Server konnte die gültige Anfrage nicht ausführen. Verantwortungsbereich des Web-Servers.

8.3.4 Aufbau einer HTTP-Nachricht-Antwort

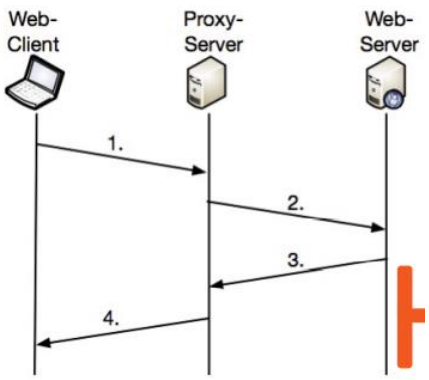
```
dig www.gmx.de # IP-Adresse Rausfinden
telnet 82.165.230.36 80 # verbinden

<Verbindungsaufbau>          #Requestanfrage
GET /index.html HTTP/1.1     #1.Methode GET, 2.Ressource 3.Protokollversion
Host: <www.gmx.de> # Host eingeben
Statuszeile <Protokollnummer> <CODE> <Erklärung>        #ANTWORTKÖRPER!!
Date: <Datum>
Server: Apache
Content-Type: text/HTML
Content-Length: 1245 #Anzahl der Zeichen des HMTL CODES
Last-Modified: Mo, 19 Oct 2016 08:00:01 GMT

<html>
<body><h1>Hello root! It works!</h1>
<p>This is the default web page for this
server.</p>
<p>The web server software is running but no
content has been added, yet.</p>
</body>
</html>
Connection closed by foreign host. #Ende
```

8.4 Das Konzept des Proxys (Anonymisierung)

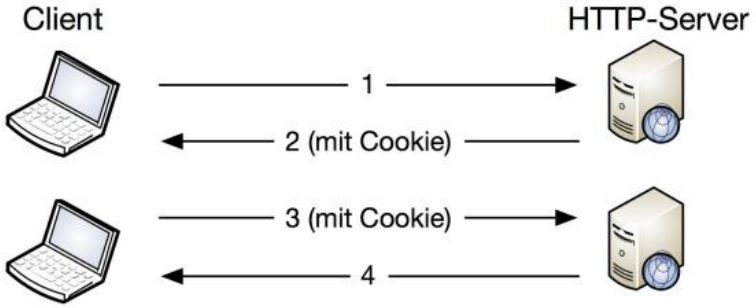
- Proxy hat eigene IP-Adresse und vermittelt zwischen Web-Client & Web-Server
- Caching: Kann daten zwischenspeichern und bei mehreren Anfragen schneller antworten
- Filtern: Kann Anfragen auswerten und verbieten
- Reverse-Proxy: das gleiche nur die Rollen vertauscht => Serverlastenkontrolle oder ähnliches



1. **Der Web-Client sendet seine Anfrage an den Proxy.**
2. **Der Proxy leitet die Anfrage an den gewünschten Web-Server weiter.**
3. **Der Web-Server sendet seine Antwort an den Proxy.**
4. **Der Proxy leitet die Antwort an den Web-Client**

8.5 Zustandsverwaltung mit Cookies

- Bei der **ersten** Anfrage sendet der Web-Server im Kopf der Antwort ein Antwort-Header-Feld, eine Cookie-Zeile: **Set-cookie: id=1678453**
  - ⇨ Web-Client schickt das Cookie im Header bei späteren Anfragen



8.6 Persistente vs. nicht persistente Verbindungen

- HTML-Seiten müssen häufig Sachen **nachladen** und bei nicht persistenten Verbindungen wird der Verbindungsaufbau nach der Antwort des Servers beendet
  - ⇨ **Überflüssig überlastet Server**
- Persistente Verbindungen bleiben bestehen und man kann einfach Sachen nachladen

HTTP/1.0	HTTP/1.1
nicht persistente Verbindungen	persistente Verbindungen

8.7 Conditional GET

- Anfrage am Server, ob ein Objekt sich verändert hat nur dann wird neu übertragen
  - ⇨ Client aktualisiert automatisch seinen Cache und guckt, ob Objekt älter ist

8.8 Bezeichnung von Dokumenten / Uniform Resource Identifier (URI)

- Uniform Resource Locator (URL): bezeichnet Ort wo dieses Dokument gespeichert ist
- Uniform Resource Name (URN): bezeichnet weltweit eindeutigen Namen für das Dokument

Aufbau:

`<scheme>://<authority><path>?<query>`

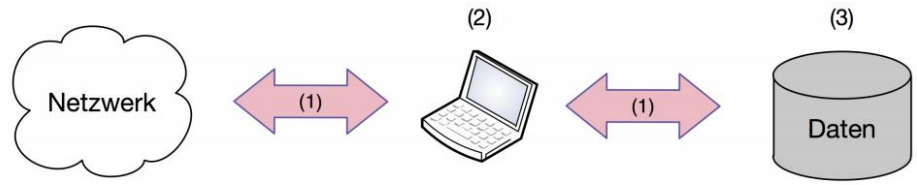
9 LZ_09_Schlüssel

9.1 Schutzziele sicherer Systeme

9.1.1 Begriffsbestimmung zur IT-Sicherheit

- Bedrohung von IT-Systeme oder Bedrohungen von Ihnen an andere Systeme
- Schaden: das Resultat von eigetretenen Bedrohungen
- Risiko: Kombination aus Wahrscheinlichkeit des Eintritts eines Schadens und Ausmaß des Schadens

9.1.2 Angriffspunkte für Bedrohungen



1. **Kommunikationswege** (z. B. Abhören von Übertragungen).
2. **Computer** (z. B. Manipulation von Rechnern).
3. **Daten** (z. B. Daten verändern, entwenden).

9.1.3 Schutzziele:

- **Vertraulichkeit**: Daten sind nur den befugten Personen zugänglich
  ⇨ durch Verschlüsselung erreicht
- **Integrität**: Daten sind korrekt und wurden während der Übertragung nicht verändert
  ⇨ Änderungen können durch kryptographische Prüfsummen erkannt werden
- **Authentizität**: Daten stammen vom vorgeblichen Erzeuger
  ⇨ Identität des Erzeugers kann durch digitale Signaturen überprüft werden
- **Verfügbarkeit**: Daten bzw. Systeme können von befugten Personen gelesen, bearbeitet oder benutzt werden
  ⇨ durch redundante Systeme erreicht

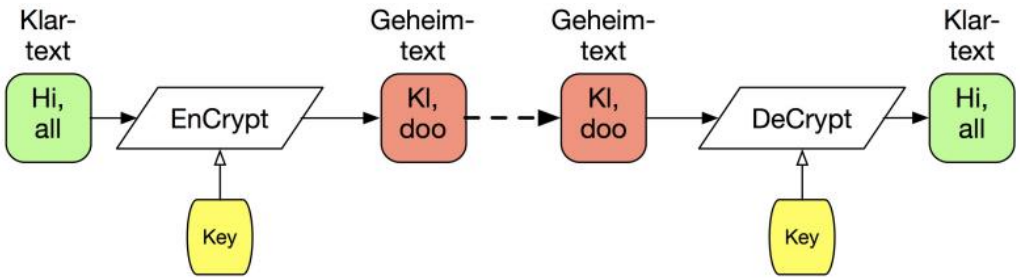
9.2 Verschlüsselungsverfahren

9.2.1 Verschlüsselung: Begriffe

- **Klartext**: ursprünglich lesbare Nachricht
- **Geheimtext**: verschlüsselte Nachricht
- **Verschlüsselung**: Verschlüsselungsalgorithmen zur Verschlüsselung der Nachricht
- **Schlüssel**: Parameter, der den Verschlüsselungsalgorithmus steuert
- **Entschlüsselung**: Umkehrung der Verschlüsselung

9.2.2 Übersicht: verschlüsselte Kommunikation

Erwartung: Ohne die Kenntnis des Schlüssels (Key) ist eine Entschlüsselung unmöglich!



9.2.3 Verschlüsselungsverfahren: Klassifikation

- **Substitution**: Zeichen werden ersetzt
- **Transposition**: Reihenfolge der Zeichen werden vertauscht
- **Anzahl der Schlüssel:**
  - ⇨ **Symmetrisch**: wenn 1 Schlüssel für das Verschlüsseln und Entschlüsseln zuständig ist (Private Key Verfahren)
  - ⇨ **Asymmetrisch**: wenn Schlüsselpaar verwendet wird (Public Key Verfahren)
- **Verarbeitung des Klartextes:**
  - ⇨ **Blockverschlüsselung**: Klartext wird in Blöcke fester Größe eingeteilt, bevor die Blöcke in „Runden" verschlüsselt werden
  - ⇨ **Stromverschlüsselung**: Klartext wird als Folge von Zeichen betrachtet und verschlüsselt jedes Zeichen einzeln

9.3 Private Key Verschlüsselung

- Symmetrische Verschlüsselung
  - ⇨ One-Time-Pad ist unknackbar

9.3.1 Cäsar-Chiffre

Chiffre(x) = (x + 3) mod 26

9.3.2 Polyalphabetische Chiffre

- monoalphabetische Substitution
- zyklisches Verschieben => Verschiebechiffre
- Schlüssel ist Zeichenkette
  - ⇨ Verschiebedistanz hängt vom Schlüssel ab
- Je länger Schlüssel desto sicherer
  - ⇨ Angreifer muss erst Schlüssellänge bestimmen
  - ⇨ Häufigkeitsanalyse durchführen

Klartext	d	a	s	i	s	t	g	e	h	e	i	m
Schlüssel	s	e	c	r	e	t	s	e	c	r	e	t
Länge	18	4	2	17	4	19	18	4	2	17	4	19
Chiffre	v	e	u	z	w	m	y	i	j	v	m	f

9.3.3 Transpositionsverschlüsselung

- Positionen verschieben
- 1 wird mir 3 vertauscht & 2 mit 4

Beispiel mit Permutation: $(1 \quad 3)(2 \quad 4)$.

Klartext	d	a	s	i	s	t	g	e	h	e	i	m
Geheimtext	s	i	d	a	g	e	s	t	i	m	h	e

9.3.4 Grundoperationen in Computern

- **Ersetzung**: im Prozessor durch die Exklusiv-Oder (XOR) Funktion $\oplus$ durchgeführt
- **Umordnung/Vertauschung:** Die Umordnung/Vertauschung wird im Prozessor durch zirkuläres Verschieben durchgeführt.

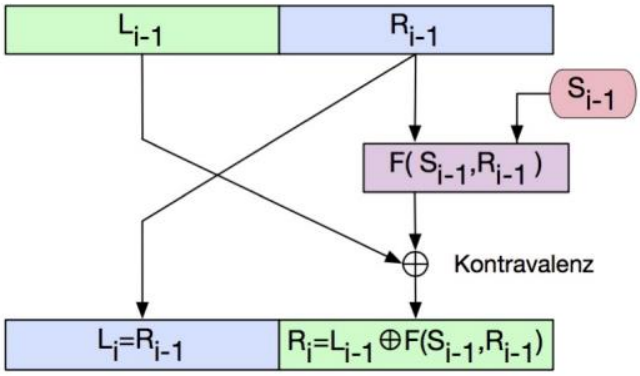
Großer Vorteil: $(a \oplus b) \oplus b = a$

a	b	$a \oplus b$	$(a \oplus b) \oplus b = a$
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

9.3.5 Feistel-Verfahren

- Architektur für Blockverschlüsselungsverfahren
  ⇨ 2 Hälften aufgeteilt und danach in Runden verarbeitet. Typische Rundenzahl: 12 bis 16

S_i sind Rundenschlüssel



9.3.6 Entschlüsselung im Feistel-Verfahren

Allgemein gilt für jede Runde i:

$$L_i = R_{i-1} \quad (1)$$

$$R_i = L_{i-1} \oplus F(S_{i-1}, R_{i-1}) \quad (2)$$

Aus Gleichung (1) folgt direkt: $R_{i-1} = L_i$
Aus Gleichung (2) folgt:

$$R_i = L_{i-1} \oplus F(S_{i-1}, R_{i-1})$$

$$R_i \oplus F(S_{i-1}, R_{i-1}) = [L_{i-1} \oplus F(S_{i-1}, R_{i-1})] \oplus F(S_{i-1}, R_{i-1}) \quad (3)$$

Wegen $(a \oplus b) \oplus b = a$ folgt aus Gleichung (3):

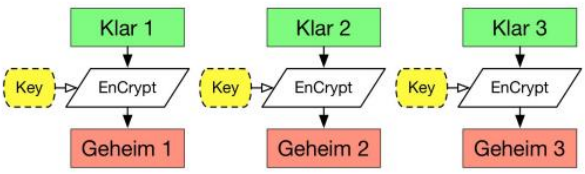
$$R_i \oplus F(S_{i-1}, R_{i-1}) = L_{i-1} \quad \text{bzw.}$$

$$L_{i-1} = R_i \oplus F(S_{i-1}, R_{i-1})$$

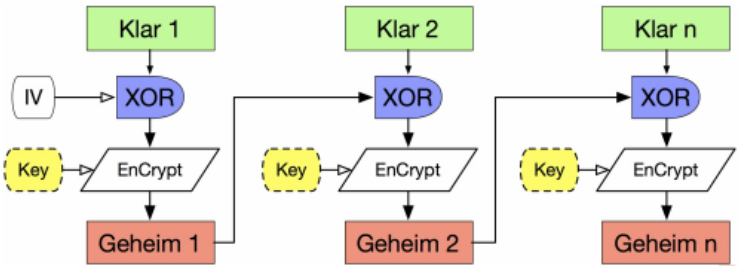
Mit Gleichung (1) folgt schliesslich: $L_{i-1} = R_i \oplus F(S_{i-1}, L_i)$
Entschlüsselung ist gleich Verschlüsselung bei umgekehrtem Einsatz der Rundenschlüssel!

9.3.7 Verschlüsselungsmodi

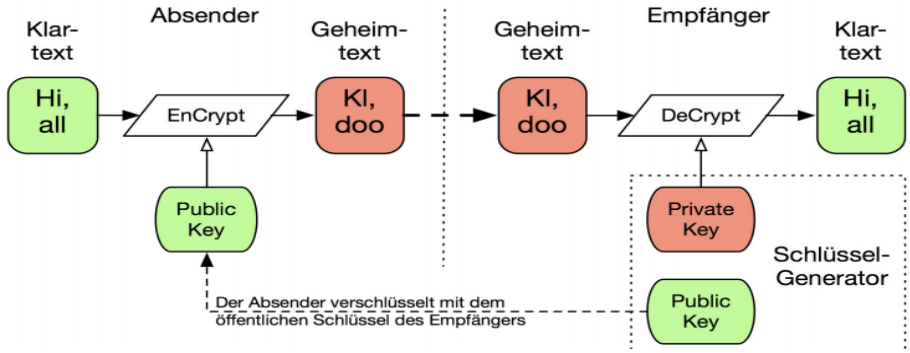
Electronic Code Book (ECB):
Jeder Klartextblock wird auf einen Geheimtextblock abgebildet
=> statistische Angriffe möglich, da Zeichen von Klar 1 & Klar 2 verschlüsselt in Geheim 1 und Geheim 2 identisch sind



Cipher Block Chaining (CBC):
Geheimtextblock entsteht aus Klartext und vorherigem Geheimtextblock (IV=Initialisierungsvektor)



- Aus dem öffentlichen Schlüssel kann man den privaten Schlüssel nicht ableiten.



9.4.1 RSA

- Zur Verschlüsselung und Erstellung digitaler Signaturen eingesetzt
- Basiert auf Zerlegung von Primfaktoren sehr großer Zahlen, extrem aufwändig
- **Nachrichten werden als große Zahlen interpretiert**

9.4.2 RSA-Algorithmus: Verschlüsselung

- **Zahlenpaar (n, e) öffentlicher Schlüssel S_{pub}**
- **Zahlentupel (n, e, d) privater Schlüssel S_{priv}**
- **n ist das Produkt von 2 Primzahlen p und q**
- Übertragene Nachricht k mit k < n & Verschlüsselte Nachricht g

$$g \equiv k^e \bmod n$$

9.4.3 RSA-Algorithmus: Entschlüsselung

- d gibt an, wie oft wir die verschlüsselte Nachricht multiplizieren müssen, dann bekommen wir die unverschlüsselte Nachricht wieder

$$g^d \equiv (k^e)^d \equiv k^{e \cdot d} \bmod n$$

$$g^d \equiv k^1 \equiv k \bmod n$$

9.4.4 Eigenschaften von RSA

- Die Sicherheit von RSA beruht darauf, dass man mit e und n nicht auf d schließen kann. Dazu müsste man p und q bestimmen, also n in seine Primfaktoren zerlegen