

Inhaltsverzeichnis

Vorwort	19
Teil I: Grundlagen	31
1 Ein- und Ausgabe-Grundlagen	33
1.1 Perl und der Netzbetrieb	33
1.1.1 Eine Sprache, wie geschaffen für Interprozess-Kommunikation	33
1.1.2 Eine Sprache, wie geschaffen für Textverarbeitung	34
1.1.3 Ein Open-Source-Projekt	34
1.1.4 Objektorientierte Netzwerk-Erweiterungen	34
1.1.5 Sicherheit	34
1.1.6 Performance	35
1.2 Netzbetrieb leicht gemacht	35
1.3 Dateihandles	37
1.3.1 Die Standard-Dateihandles	37
1.3.2 Ein- und Ausgabeoperationen	38
1.3.3 Das Dateiende finden	41
1.3.4 Dateien öffnen und schließen	44
1.3.5 Puffern und Blockieren	49
1.3.6 Dateihandles übergeben und speichern	52
1.3.7 Fehler finden	54
1.3.8 Die objektorientierte Syntax der Module IO::Handle und IO::File verwenden	55
1.4 Zusammenfassung	61
2 Prozesse, Pipes und Signale	63
2.1 Prozesse	63
2.1.1 Die Funktion fork()	64
2.1.2 Die Funktionen system() und exec()	66
2.2 Pipes	67
2.2.1 Eine Pipe öffnen	68
2.2.2 Pipes verwenden	68
2.2.3 Pipes leicht gemacht: Der Backtick-Operator	70
2.2.4 Pipes mit mehr Möglichkeiten: Die Funktion pipe()	71

2.2.5	Bidirektionale Pipes	74
2.2.6	Zwischen Pipes und einfachen Dateihandles unterscheiden	75
2.2.7	Der gefürchtete PIPE-Fehler	75
2.3	Signale	77
2.3.1	Gebräuchliche Signale	77
2.3.2	Signale abfangen	79
2.3.3	Behandlung von PIPE-Ausnahmen	81
2.3.4	Signale senden	83
2.3.5	Beachtenswertes über Signalhandler	84
2.3.6	Timeout langsamer Systemaufrufe	85
2.4	Zusammenfassung	86
3	Einführung in Berkeley Sockets	89
3.1	Clients, Server und Protokolle	89
3.1.1	Protokolle	89
3.1.2	Binäre Protokolle im Vergleich zu textorientierten Protokollen	91
3.2	Berkeley Sockets	92
3.2.1	Der Aufbau eines Sockets	93
3.2.2	Datagramm-Sockets	95
3.2.3	Stream-Sockets	96
3.2.4	Datagramme kontra Stream-Sockets	97
3.3	Socket-Adressierung	98
3.3.1	IP-Adressen	98
3.3.2	Reservierte IP-Adressen, Subnetze und Netzmasken	99
3.3.3	IPv6	101
3.3.4	Netzwerk-Ports	101
3.3.5	Die sockaddr_in-Struktur	102
3.4	Ein einfacher Netzwerk-Client	103
3.5	Netzwerknamen und -dienste	105
3.5.1	Hostnamen in IP-Adressen umwandeln	105
3.5.2	Beispiele für die Hostnamen-Umwandlung	106
3.5.3	Informationen über Protokolle und Dienste erhalten	107
3.5.4	Der verbesserte Daytime-Client	108
3.5.5	Andere Quellen für Netzwerkinformationen	109
3.6	Netzwerkanalyse-Tools	110
3.6.1	ping	110
3.6.2	nslookup	111

3.6.3	traceroute	111
3.6.4	netstat	112
3.6.5	tcpdump	113
3.6.6	MacTCP Watcher	113
3.6.7	scanner.exe	113
3.6.8	net-toolbox.exe	113
3.7	Zusammenfassung	113
4	Das TCP-Protokoll	115
4.1	Ein TCP-Echo-Client	115
4.2	Socket-Funktionen zur Bearbeitung ausgehender Verbindungen	118
4.3	Ein TCP-Echo-Server	119
4.3.1	Socket-Funktionen zur Bearbeitung ankommender Verbindungen	124
4.3.2	Einschränkungen von tcp_echo_serv1.pl	125
4.4	Socket-Optionen anpassen	126
4.4.1	Gängige Socket-Optionen	127
4.4.2	Die Socket-Option SO_REUSEADDR	128
4.4.3	Die Funktionen fcntl() und ioctl()	129
4.5	Weitere Socket-bezogene Funktionen	129
4.5.1	Vom Modul Socket exportierte Zeilenende-Konstanten	130
4.6	Ausnahmezustände während der TCP-Kommunikation	131
4.6.1	Ausnahmen während des Verbindens	131
4.6.2	Ausnahmen bei Lese- und Schreiboperationen	132
4.7	Zusammenfassung	133
5	Die IO::Socket-API	135
5.1	IO::Socket verwenden	135
5.1.1	Ein Daytime-Client	135
5.1.2	TCP-Echo-Client	136
5.2	IO::Socket-Methoden	138
5.2.1	Die IO::Handle-Klassenhierarchie	138
5.2.2	IO::Socket::INET-Objekte erzeugen	138
5.2.3	IO::Socket-Objektmethoden	142
5.3	Praktischere Beispiele	144
5.3.1	Der verbesserte Echo-Umkehr-Server	145
5.3.2	Ein Web-Client	147
5.4	Performance und Stil	150

Inhaltsverzeichnis

5.5	Zusammentreffen von Clients	151
5.5.1	Ein Dialog-Client, erster Versuch	151
5.5.2	Ein Dialog-Client, zweiter Versuch	153
5.6	Zusammenfassung	158
Teil II:	Clients für gängige Dienste entwickeln	159
6	FTP und Telnet	161
6.1	Net::FTP	161
6.1.1	Ein Net::FTP-Beispiel	161
6.1.2	FTP und befehlsorientierte Protokolle	163
6.1.3	Die Net::FTP-API	165
6.1.4	Ein Verzeichnis-Mirror-Skript	168
6.2	Net::Telnet	176
6.2.1	Ein einfaches Net::Telnet-Beispiel	177
6.2.2	Die Net::Telnet-API	178
6.2.3	Ein Programm zur Änderung entfernter Passwörter	182
6.2.4	Net::Telnet mit Nicht-Telnet-Protokollen verwenden	186
6.2.5	Das Modul Expect	191
6.3	Zusammenfassung	192
7	SMTP: Mail versenden	193
7.1	Einführung in die Mail-Module	193
7.2	Net::SMTP	193
7.2.1	Das SMTP-Protokoll	194
7.2.2	Die Net::SMTP-API	195
7.2.3	Net::SMTP verwenden	198
7.3	MailTools	200
7.3.1	MailTools verwenden	200
7.3.2	Mail::Header	201
7.3.3	Mail::Internet	204
7.3.4	Ein Mail-Autoreply-Programm	206
7.3.5	Mail::Mailer	209
7.4	MIME-Tools	211
7.4.1	Eine kurze Einführung in MIME	211
7.4.2	Organisation der MIME::*-Module	215
7.4.3	MIME::Entity	217

7.4.4	MIME::Head	223
7.4.5	MIME::Body	223
7.4.6	MIME::Parser	225
7.4.7	MIME-Beispiel: Die neuesten CPAN-Einträge versenden	230
7.5	Zusammenfassung	234
8	POP, IMAP und NNTP – Mail und Netnews verarbeiten	235
8.1	Das Post Office Protocol	235
8.1.1	Übersicht einer POP3-Mailbox erstellen	235
8.1.2	Net::POP3-API	238
8.1.3	MIME-Nachrichten mittels POP abholen und verarbeiten	240
8.1.4	Das Skript pop_fetch.pl	242
8.1.5	Das Modul PopParser	250
8.2	Das IMAP-Protokoll	252
8.2.1	Übersicht einer IMAP-Mailbox erstellen	253
8.2.2	Die Net::IMAP::Simple-API	255
8.3	Internet News Clients	258
8.3.1	Net::NNTP	260
8.3.2	Die Net::NNTP-API	262
8.4	Ein News-Mail-Gateway	269
8.5	Zusammenfassung	278
9	Web-Clients	279
9.1	LWP installieren	279
9.2	LWP-Grundlagen	281
9.2.1	HTTP::Request	283
9.2.2	HTTP::Response	287
9.2.3	LWP::UserAgent	290
9.3	LWP-Beispiele	295
9.3.1	Eine Liste von RFCs holen	295
9.3.2	Eine Liste von RFCs spiegeln	296
9.3.3	Formulare simulieren	298
9.3.4	HTTP::Request::Common für den Formularversand benutzen	303
9.3.5	Datei-Uploads mit Hilfe von multipart/form-data	305
9.3.6	Eine passwortgeschützte Seite abholen	310
9.4	Parsing von HTML und XML	312
9.4.1	HTML formatieren	313

9.4.2	Die HTML::Formatter-API	314
9.4.3	Die HTML::TreeBuilder-API	315
9.4.4	Formatiertes HTML aus dem Skript get_url.pl zurückgeben	317
9.4.5	Das Modul HTML::Parser	319
9.4.6	HTML::Parser verwenden	320
9.4.7	Die HTML::Parser-API	322
9.4.8	search_rfc.pl unter Verwendung von HTML::Parser	324
9.4.9	Bilder von einem entfernten URL extrahieren	327
9.5	Zusammenfassung	331
Teil III: TCP-Client/Server-Systeme entwickeln		333
10	Forking-Server und der inetd-Daemon	335
10.1	Standardverfahren für die Gleichzeitigkeit	335
10.1.1	Forking-Server	335
10.1.2	Multithreading-Server	336
10.1.3	Multiplex-Server	337
10.2	Funktionierendes Beispiel: Ein Psychotherapie-Server	337
10.3	Der Psychotherapeut als Forking-Server	338
10.3.1	Zombies	339
10.3.2	Reaping von Child-Prozessen im CHLD-Handler	340
10.3.3	Psychotherapie-Server mit Forking	341
10.3.4	Den Psychotherapie-Server auf der Windows-Plattform verwenden	345
10.4	Ein Client-Skript für den Psychotherapie-Server	345
10.5	UNIX-Server zu Daemons machen	348
10.5.1	Automatischer Wechsel in den Hintergrund	349
10.5.2	PID-Dateien	350
10.6	Netzwerkserver automatisch starten	355
10.6.1	Hintergrundprozesse auf Windows- und Macintosh-Systemen	357
10.7	Den inetd-Superdaemon benutzen	358
10.7.1	inetd einsetzen	360
10.7.2	inetd im wait-Modus verwenden	362
10.8	Zusammenfassung	365
11	Multithreading-Anwendungen	367
11.1	Über Threads	367
11.1.1	Threads sind experimentell	367
11.1.2	Die Thread-API	368

11.1.3	Eine einfache Multithreading-Anwendung	369
11.1.4	Sperren	369
11.1.5	Thread-Modulfunktionen und -Methoden	372
11.1.6	Threads und Signale	374
11.2	Ein Multithreading-Psychiatrie-Server	374
11.2.1	Die Klasse Chatbot::Eliza::Server	376
11.3	Ein Multithreading-Client	377
11.4	Zusammenfassung	379
12	Multiplex-Anwendungen	381
12.1	Ein Multiplex-Client	381
12.2	Das Modul IO::Select	383
12.2.1	Die eingebaute Funktion select()	385
12.2.2	Wann ist ein Dateihandle I/O-bereit?	385
12.2.3	select() mit Standard-I/O kombinieren	387
12.2.4	Die »Niedrigwassermarken« anpassen	387
12.3	Ein Multiplex-Psychiatrie-Server	387
12.3.1	Das Server-Hauptprogramm	388
12.3.2	Das Modul Chatbot::Eliza::Polite	391
12.3.3	Probleme mit dem Psychiatrie-Server	393
12.3.4	Win32-Probleme	394
12.4	Zusammenfassung	394
13	Nicht blockierende Ein- und Ausgabe	395
13.1	Nicht blockierende I/O-Handles erzeugen	395
13.1.1	Nicht blockierende Handles erzeugen: Die Funktionsschnittstelle	396
13.1.2	Nicht blockierende Handles erzeugen: Die objektorientierte Schnittstelle	397
13.2	Nicht blockierende Handles verwenden	398
13.2.1	sysread() mit nicht blockierenden Dateihandles	398
13.2.2	syswrite() mit nicht blockierenden Dateihandles	399
13.3	Nicht blockierende Handles mit zeilenorientiertem I/O verwenden	400
13.3.1	IO::Getline verwenden	402
13.3.2	Das Modul IO::Getline	403
13.4	Ein allgemeingültiges, nicht blockierendes I/O-Modul	407
13.4.1	Ein nicht blockierender Echo-Server	407
13.4.2	Ein nicht blockierender, zeilenorientierter Server	409

13.4.3	Das Modul IO::SessionData	412
13.4.4	Das Modul IO::SessionSet	421
13.4.5	Die Klassen IO::LineBufferedSet und IO::LineBufferedSessionData	428
13.4.6	IO::SessionSet mit Handles verwenden, die keine Sockets sind	430
13.5	Nicht blockierende connect()- und accept()-Aufrufe	433
13.5.1	Der IO::Socket-Parameter Timeout	433
13.5.2	Nicht blockierendes connect()	434
13.5.3	Mehrere gleichzeitige connect()-Aufrufe	437
13.5.4	Ein einfacher HTTP-Client	437
13.5.5	Das Modul HTTPFetch	441
13.5.6	Nicht blockierende accept()-Aufrufe	447
13.6	Zusammenfassung	448
14	Absichern von Servern	449
14.1	Das System Log verwenden	449
14.1.1	Über UNIX-Syslog	450
14.1.2	Sys::Syslog	452
14.1.3	Logging zum Psychotherapie-Server hinzufügen	453
14.1.4	Logging mit warn() und die()	459
14.1.5	Event Log auf der Win32-Plattform verwenden	460
14.1.6	Direktes Logging in eine Datei	461
14.2	Benutzerprivilegien setzen	465
14.2.1	User- und Group-ID ändern	465
14.2.2	Den Psychotherapie-Server als Root starten	467
14.3	Der Taint-Modus	470
14.3.1	Den Taint-Modus verwenden	472
14.4	chroot() verwenden	473
14.4.1	chroot() zum Psychotherapie-Server hinzufügen	473
14.5	Die Verarbeitung von HUP und anderen Signalen	475
14.5.1	Änderungen am Haupt-Skript	476
14.5.2	Änderungen am Modul Daemon	479
14.6	Zusammenfassung	486
15	Preforking und Prethreading	489
15.1	Preforking	489
15.1.1	Ein Webserver	491
15.1.2	Ein Webserver, der Anfragen nacheinander verarbeitet	497
15.1.3	Ein Webserver mit Akzeptieren und Forking	497

15.1.4	Der Preforking-Webserver, Version 1	500
15.1.5	Der Preforking-Webserver, Version 2	502
15.1.6	Ein anpassungsfähiger Preforking-Server	506
15.1.7	Ein anpassungsfähiger Preforking-Server, der gemeinsamen Speicher verwendet	514
15.2	Prethreading	521
15.2.1	Ein Threading-Webserver	521
15.2.2	Ein einfacher Prethreading-Server	523
15.2.3	Anpassungsfähiges Prethreading	524
15.2.4	Das Modul NetServer::Generic	529
15.3	Performance-Messungen	530
15.4	Zusammenfassung	531
16	IO::Poll	533
16.1	IO::Poll verwenden	533
16.2	IO::Poll-Ereignisse	535
16.2.1	IO::Poll-Methoden	536
16.2.2	Ein nicht blockierender TCP-Client, der IO::Poll verwendet	537
16.3	Zusammenfassung	541
Teil IV:	Fortgeschrittene Themen	543
17	TCP Urgent Data	545
17.1	»Out-of-Band«-Daten und der Urgent-Zeiger	545
17.2	TCP Urgent Data verwenden	547
17.2.1	Die Option SO_OOBLINE	550
17.2.2	select() mit Urgent Data verwenden	552
17.3	Die Funktion sockatmark()	553
17.3.1	sockatmark() implementieren	554
17.4	Ein Travesty-Server	556
17.4.1	Das Modul Text::Travesty	557
17.4.2	Das Travesty-Server-Design	558
17.4.3	Der Travesty-Client	564
17.4.4	Den Travesty-Server testen	569
17.4.5	Das Modul IO::Sockatmark	570
17.5	Zusammenfassung	570

18 Das UDP-Protokoll	573
18.1 Ein Daytime-Client	573
18.2 UDP-Sockets erzeugen und verwenden	576
18.2.1 UDP-Socket-Erzeugung	576
18.3 Die Funktionen send() und recv()	576
18.3.1 Ein UDP-Socket binden	577
18.3.2 Ein UDP-Socket »verbinden«	577
18.4 UDP-Fehler	578
18.4.1 Asynchrone Fehler	578
18.4.2 Ausgelassene Pakete und Fragmentierung	579
18.5 UDP-Sockets mit IO::Socket verwenden	579
18.5.1 Daytime-Client, der IO::Socket verwendet	580
18.6 Senden an mehrere Hosts	581
18.7 UDP-Server	584
18.7.1 Ein UDP-Umkehr-Echo-Server	585
18.7.2 UDP-Echo-Client	586
18.8 Die Stabilität von UDP-Anwendungen steigern	588
18.8.1 Timeout beim Empfang von UDP	589
18.8.2 Duplikate und Datagramme in der falschen Reihenfolge	591
18.9 Zusammenfassung	597
 19 UDP-Server	 599
19.1 Ein Internet-Chat-System	599
19.1.1 Eine Beispiel-Sitzung	599
19.1.2 Chat-System-Aufbau	601
19.2 Der Chat-Client	603
19.2.1 Das Modul ChatObjects::Comm	611
19.2.2 Das Modul ChatObjects::ChatCodes	613
19.3 Der Chat-Server	614
19.3.1 Das Hauptskript des Servers	614
19.3.2 Die Klasse ChatObjects::User	617
19.3.3 Die Klasse ChatObjects::Channel	623
19.4 Beendete Clients ausfindig machen	626
19.4.1 STILL_HERE-Ereignisse zum Chat-System hinzufügen	627
19.4.2 Änderungen an ChatObjects::ChatCodes	627
19.4.3 Die Unterklasse ChatObjects::TimedUser	628

19.4.4	Das geänderte Programm chat_client.pl	628
19.4.5	Das geänderte Programm chat_server.pl	630
19.5	Zusammenfassung	633
20	Broadcasting	635
20.1	Unicasting kontra Broadcasting	635
20.2	Broadcasting näher erklärt	636
20.2.1	Broadcast-Anwendungen	637
20.3	Broadcasts senden und empfangen	637
20.3.1	Broadcasts senden	638
20.3.2	Broadcasts empfangen	641
20.4	Broadcasting ohne Broadcast-Adresse	642
20.4.1	Die »Nur-Einser«-Broadcast-Adresse	642
20.4.2	Broadcast-fähige Schnittstellen während der Laufzeit ermitteln	643
20.4.3	Das Modul IO::Interface	644
20.4.4	Schrittweise Erläuterung von IO::Interface	646
20.5	Den Chat-Client mit der Fähigkeit der Ressourcen-Findung ausstatten	654
20.6	Zusammenfassung	656
21	Multicasting	659
21.1	Multicasting-Grundlagen	659
21.1.1	Reservierte Multicast-Adressen	660
21.1.2	Multicast-Adressen und Hardwarefilterung	661
21.1.3	Multicasting durch WANs	662
21.1.4	Multicast-TTLs	663
21.2	Multicast verwenden	665
21.2.1	Multicast-Nachrichten versenden	665
21.2.2	Socket-Optionen für den Multicast-Versand	666
21.2.3	Multicast-Nachrichten erhalten	668
21.2.4	Das Modul IO::Socket::Multicast	669
21.3	Multicast-Beispielanwendungen	674
21.3.1	Multicasting-Daytime-Server	675
21.3.2	Multicast-Daytime-Client	676
21.3.3	Multicast-Chat-System	678
21.3.4	Zusammenfassung	690

22 UNIX-Domain-Sockets	691
22.1 UNIX-Domain-Sockets verwenden	691
22.1.1 Die funktionsorientierte Schnittstelle für UNIX-Domain-Sockets	692
22.1.2 Die objektorientierte Schnittstelle für UNIX-Domain-Sockets	694
22.1.3 UNIX-Domain-Sockets und Dateirechte	695
22.2 Ein »Umbruch«-Server	696
22.2.1 Der Text::Wrap-Server	696
22.2.2 Der Text::Wrap-Client	699
22.3 Die Verwendung von UNIX-Domain-Sockets für Datagramme	700
22.3.1 UNIX-Domain-Daytime-Server	701
22.3.2 UNIX-Domain-Daytime-Client	703
22.4 Zusammenfassung	705
 Anhang	
 A Zusätzlicher Quellcode	707
A.1 Net::NetmaskLite (Kapitel 3)	707
A.2 PromptUtil.pm (Kapitel 8 und 9)	710
A.3 IO::LineBufferedSet (Kapitel 13)	714
A.4 IO::LineBufferedSessionData (Kapitel 13)	717
A.5 DaemonDebug (Kapitel 14)	725
A.6 Text::Travesty (Kapitel 17)	727
A.7 mchat_client.pl (Kapitel 21)	731
 B Perl-Fehlercodes und spezielle Variablen	737
B.1 Systemfehler-Konstanten	737
B.2 Magische Variablen, die I/O betreffen	741
B.3 Weitere globale Perl-Variablen	742
 C Internet-Referenztabellen	745
C.1 Zugewiesene Port-Nummern	745
C.2 Registrierte Portnummern	766
C.3 Internet-Multicast-Adressen	784

D Bibliographie	787
D.1 Perl-Programmierung	787
D.1.1 Bücher	787
D.1.2 Online-Ressourcen	787
D.2 TCP/IP und Berkeley Sockets	787
D.2.1 Bücher	787
D.2.2 Online-Ressourcen	788
D.3 Netzwerk-Server-Design	788
D.4 Multicasting	788
D.4.1 Bücher	788
D.4.2 Online-Ressourcen	789
D.5 Anwendungsprotokolle	789
D.5.1 FTP	789
D.5.2 Telnet	789
D.5.3 Secure Shell	789
D.5.4 SMTP	790
D.5.5 MIME	790
D.5.6 POP	790
D.5.7 IMAP	791
D.5.8 NNTP	791
D.5.9 HTTP, HTML und XML	791
D.5.10 Netzwerk-Sicherheit	792
 Stichwortverzeichnis	 793