

Grundlagen Betriebssysteme und Rechnernetze

Wintersemester 2018/19

Inhaltsverzeichnis

1 Einleitung	5	2.4.5 Beispiel: Java-Threads	23
1.1 Definition eines Betriebssystems	5	2.5 Kommunikation	23
1.2 Aufgaben eines Betriebssystems	6	2.5.1 IPC - Signale	23
1.2.1 Prozessverwaltung	6	2.5.2 IPC - Pipes	25
1.2.2 Speicherverwaltung	8	2.5.3 IPC - Gemeinsamer Speicher (Shared Memory)	27
Grundlagen Betriebssysteme	5	2.5.4 IPC - Nachrichten: Sockets	27
2 Prozessverwaltung	9	2.5.5 Übersicht über die Socket-Schnittstelle	28
2.1 Einführung	9	3 Synchronisation nebenläufiger Prozesse	34
2.1.1 Abnormales Prozessverhalten	10	3.1 Synchronisationshardware	34
2.1.2 Unterbrechungen	10	3.2 Busy Waiting	34
2.1.3 Erzeugen von Prozessen	11	3.2.1 Lösungsverfahren für 2 Prozesse: Alternierender Zutritt	34
2.1.4 Beenden von Prozessen	12	3.2.2 Peterson's Algorithmus (1981)	35
2.2 Scheduling	13	3.3 Spin Locking mit Hardwareunterstützung	36
2.2.1 First-In-First-Out (FIFO)	15	3.4 Semaphor	38
2.2.2 Shortest Job First (SJF)	16	3.4.1 Producer-Consumer Problem	39
2.2.3 Prioritäten-basiertes Scheduling	16	3.4.2 Readers-Writers Problem	40
2.2.4 Betriebssystemkomponenten	17	3.4.3 Nachteile von Semaphoren	42
2.2.5 Beispiel: Ablaufsteuerung unter UNIX	17	3.5 Monitor	42
2.3 Beispiele	18	3.5.1 Monitorlösung für Producer-Consumer Problem	42
2.3.1 4.3 BSD UNIX Scheduler	18	3.5.2 Implementation von Monitoren	43
2.3.2 Linux	19	3.5.3 Thread-Synchronisation in Java: Variante von Hoare's Monitorkonzept	43
2.3.3 Red-Black-Tree (<code>rbtree.h</code>)	20	3.6 Beispiel Pthread-Bibliothek	43
2.3.4 Funktionsweise CFS - Überblick	20	3.6.1 Pthread Mutex	43
2.3.5 Echtzeit-Scheduling unter Linux	21	3.6.2 Pthread Condition	46
2.4 Thread-Systeme	22		
2.4.1 Kernel-Level-Threads	22		
2.4.2 User-Level-Threads	22		
2.4.3 Thread Safety	22		
2.4.4 Parallelisierungskonzepte	23		

3.6.3	Pthread Read-Write-Locks	51		
3.7	Nachrichtensystem	51	7.2	Fehlererkennung und Fehlerkorrektur
4	Speicherverwaltung	54	7.2.1	Fehlerkorrekturcodes . . .
4.1	Swapping	54	7.2.2	Beispiel für 1-Bit-Korrekturcode (Hamming 1950)
4.1.1	Belegungsstrategien	54	7.2.3	Fehlererkennung mittels Cyclic Redundancy Code (CRC)
4.1.2	Nachteil von Swapping . .	55	7.2.4	Algorithmus zur Berechnung des Codeworts
4.1.3	Lokalitätsprinzip	55	7.2.5	Bemerkungen
4.2	Virtuelle Adressierung	55	7.3	Flußsteuerung
4.3	Paging-Verfahren	55	7.3.1	Kommunikationsarten zwischen Prozessen A, B .
4.3.1	Ersetzungsstrategien	56	7.3.2	Stop-and-Wait-Protokolle .
4.3.2	Lokalitätsprinzip für Demand-Paging	57	7.3.3	PAR-Protokolle (Positive Acknowledgement with Retransmission)
4.4	[BEISPIEL: UNIX!]	58	7.3.4	Protokollschwächen
4.5	[BEISPIEL: WINDOWS NT!]	58	7.3.5	Schiebefenster-Protokolle .
4.6	[BEISPIEL: LINUX!]	58	7.3.6	1. Möglichkeit: Go-Back-N
5	Dateisysteme	59	7.3.7	2. Möglichkeit: Selective Repeat
5.1	Benutzerschnittstelle	60	7.4	High-level Data Link Control (HDLC)
5.2	Implementation	62	7.4.1	Verbindungs-/Betriebsarten .
5.3	Berkeley Fast File System	63	7.4.2	nichtbalancierte Konfiguration
5.4	Linux Dateisystem	64	7.4.3	balancierte Konfiguration .
5.5	Windows 2000: NTFS	66	7.4.4	Rahmenformat
5.6	Sicherheit	67	7.4.5	Rahmentypen
I	Grundlagen Rechnernetze	70	7.4.6	Schiebefensterprotokoll .
6	Das ISO-Referenzmodell OSI	114	7.5	PPP – Point-to-Point-Protocol .
6.1	Bitübertragungsschicht (Physical Layer)	114	7.5.1	PPP Rahmenformat
6.2	Sicherungsschicht (Link Layer) .	114	7.5.2	Link Control Protocol (LCP)
6.2.1	Aufgaben	114	7.5.3	Aufgaben
6.3	Vermittlungsschicht (Network Layer)	115	7.5.4	Die LCP-Rahmentypen .
6.3.1	Aufgaben	115	7.5.5	Network Control Protocol (NCP)
6.3.2	Dienstarten	115	8	Der IEEE-Standard 802 für lokale Netze
6.4	Transportschicht (Transport Layer)	116	8.1	Ethernet
6.5	Sitzungsschicht (Session Layer) .	116	8.1.1	Ethernet Paketformat (1983-1996)
6.5.1	Beispiele für OSI-Dienste .	116		
6.6	Darstellungsschicht (Presentation Layer)	116		
6.7	Anwendungsschicht (Application Layer)	116		
7	Die Sicherungsschicht	117		
7.1	Aufgaben	117		
7.1.1	Zusammenstellen der Rahmen	117		
7.1.2	Fehlerkontrolle	118		
7.1.3	Flußsteuerung	118		

8.1.2	CSMA/CD-Verfahren (Carrier Sense Multiple Access with Collision Detection)	125	II Hausaufgaben	132
8.1.3	Binary Exponential Back-off Algorithm	125	1 Blatt	132
8.1.4	Eigenschaften CSMA/CD	125	1.1 Unix-Stammbaum	132
8.2	Digital Subscriber Line (DSL) . .	125	1.2 Betriebssysteme	132
8.3	PPP over Ethernet (PPPoE) . . .	126	1.3 Prozessverwaltung	132
	8.3.1 Discovery-Stage	126	1.4 Prozesstabelle	133
	8.3.2 Session-Stage	126	1.5 Prozesszustände	133
8.4	VLAN-Technologie mittels Ethernet	126	1.6 Deadlocks	134
	8.4.1 802.1Q Tag Format	126	1.7 Fork	134
9	Die Vermittlungsschicht	127	2 Blatt	136
9.1	Wegwahlverfahren (Routing) und Addressierung	127	2.1 Prozesserzeugung	136
	9.1.1 Ziele eines Wegwahlverfahrens	127	2.5 POSIX-Threads	136
	9.1.2 Klassifikation von Wegwahlverfahren	127	2.2 Echtzeit-Scheduling	136
	9.1.3 Link-State-Routing-Algorithmus	128	2.3 Linux Scheduler	137
	9.1.4 [DIJKSTRA-ALGORITHMUS!]	128	2.4 Windows Scheduler	139
	9.1.5 Hierarchische Addressierung	128	3 Blatt	141
	9.1.6 Beispiele für adaptive Verfahren	128	3.1 IPC - Shared Memory	141
	9.1.7 1. Hot-Potato-Routing . .	128	3.2 Interprozesskommunikation	142
	9.1.8 2. Backward Learning . .	128	3.3 IPC - Signale	142
	9.1.9 3. Vector Distance Routing	128	3.4 Synchronisation: gemeinsame Variable	143
9.2	Fragmentierung	129	3.5 Banker's Algorithmus	144
	9.2.1 Transparente Fragmentierung	129	4 Blatt	147
	9.2.2 Internetfragmentierung . .	129	4.1 Fork - Threads	147
9.3	Internet Protocol (IP)	130	4.2 Interprozesskommunikation	147
	9.3.1 Kennzeichen	130	4.3 Atomare Operationen	148
	9.3.2 IP Forward-Algorithm: RouteDatagram (Datagram, Wegwahltablelle) . .	130	4.4 Synchronisation	148
	9.3.3 IP Packet Processing - Outgoing Packet	130	4.5 Synchronisation nebenläufiger Prozesse	149
	9.3.4 IP Wegwahltabellen	130	5 Blatt	153
	9.3.5 Internet Control Message Protocol (ICMP)	131	5.1 Synchronisation von Threads	153
	9.3.6 Beispiel	131	5.2 POSIX-Threads, Reader/Writer Locks	153
			5.3 Speicherverwaltung	156
			5.4 Speicherverwaltung	159
			5.5 Speicherverwaltung	159
			6 Blatt	162
			6.1 POSIX-Threads, verkettete Liste, Master-Worker-Parallelisierungskonzept	162
			6.2 Verwaltung virtueller Adressraum	162
			6.3 Dateisystem	162
			6.4 Dateisystem	163
			6.5 I/O-Scheduling	163
			7 Blatt	165
			7.1 Fehlererkennung	165
			7.2 Rechnernetze: CRC	166

7.3	Rechnernetze: Leitungsauslastung	168	1.2	Prozessankunft: FCFS, SJF, RR .	169
7.4	Sicherungsschicht	168	1.3	Signale Pipes Sockets	171
III	Klausuren	169	1.4	Seitenzugriffe: BO FIFO	172
1	Klausur 2010	169	1.5	Daten senden, HDLC-Frame . . .	173
1.1	Prozess, Nebenläufigkeit, fork()	. 169	1.6	Semaphor Fahrstuhl	174
				Stichwortverzeichnis	176

1 Einleitung

1.1 Definition eines Betriebssystems

Definition 1. Das **Betriebssystem** wird durch die Programme eines digitalen Rechensystems gebildet, die zusammen mit den Eigenschaften der Rechenanlage die Basis der möglichen Betriebsarten des digitalen Rechensystems bilden und insbesondere die Abwicklung von Programmen steuern und überwachen.

Betriebsarten:

- Einprogramm-/Mehrprogrammbetrieb (single/multi programming)
- Stapelbetrieb / Dialogbetrieb (batch/interactive mode)

Definition 2. Ein **Betriebssystem** ist eine **virtuelle Maschine**, die dem Anwender eine einfache (dateiorientierte) Schnittstelle zur Hardware zur Verfügung stellt und eine Programmierung dieser Hardware auf hohem logischen Niveau ermöglicht

Compiler, Editor, Spiele, usw.
Komandointerpreter (Shell), Graphische Oberfläche
Betriebssystem
Machienenprogrammebene
Mikroprogrammebene
physikalisches Gerät

Abbildung 1: Schichtenmodell eines Betriebssystems

Betriebssystem als Ressourcenverwalter

Definition 3. Die **Ressourcen (Betriebsmittel)** eines Betriebssystems sind alle Hard- und Softwarekomponenten, die für die Programmausführung relevant sind.

Betriebsmittel sind zum Beispiel Prozessor, Hauptspeicher, Ein-/Ausabegeräte, Daten, ausführbarer Code, usw.

Definition 4. Ein **Betriebssystem** bezeichnet alle Softwarekomponenten eines Rechensystems, die die Ausführung der Benutzerprogramme, die Verteilung der Ressourcen auf die Benutzerprogramme und die Aufrechterhaltung der Betriebsart steuern und überwachen.

1.2 Aufgaben eines Betriebssystems

1.2.1 Prozessverwaltung

Definition 5. Ein **Prozess** (**process, task**) ist ein in Ausführung befindliches Programm bestehend aus

- einer Folge von Maschinenbefehlen, die durch das ausgeführte Programm festgelegt sind (**program code, text section**),
- dem Inhalt des Stapel-Speichers (**stack**), auf dem temporäre Variablen und Parameter für Funktionsaufrufe verwaltet werden,
- dem Inhalt des Speichers, in dem die globale Daten des Prozesses gehalten werden **data section** und
- Verwaltungsinformationen, die den aktuellen **Bearbeitungszustand (Kontext)** beschreiben.

Der Programmkontext wird u.a. durch den Programmzähler und die Registerinhalte des Prozessors beschrieben (**internal state**).

Definition 6. Die Laufzeit eines Prozesses P wird durch ein Tupel $(a_1, a_2) \in \mathbb{R}^2$ beschrieben, wobei $a_1 =$ Startzeit von P und $a_2 =$ die Endzeit von P ist.

Definition 7. A und B seien Prozesse mit Laufzeit (a_1, a_2) bzw. (b_1, b_2) .

- A läuft vor B (i.Z. $A \rightarrow B$) $\Leftrightarrow a_2 < b_1$.
- A und B heißen **sequentielle Prozesse** $\Leftrightarrow A \rightarrow B \vee B \rightarrow A$
- A und B heißen **nebenläufig** $\Leftrightarrow [a_1, a_2] \cap [b_1, b_2] \neq \emptyset$
- A und B heißen **parallel** \Leftrightarrow Es existiert ein Zeitintervall, während dessen A und B gleichzeitig rechnen.

Bemerkung

1. " \rightarrow " irreflexive Halbordnung

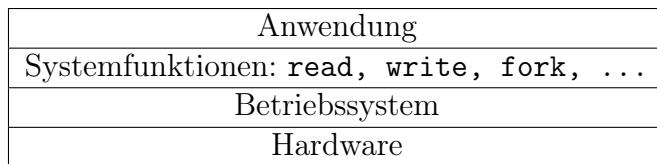
$$\begin{aligned} A \rightarrow B \Rightarrow B \not\rightarrow A && (\text{asymmetrisch}) \\ A \rightarrow B \wedge B \rightarrow C \Rightarrow A \rightarrow C && (\text{transitiv}) \\ A \not\rightarrow A && (\text{irreflexiv}) \end{aligned}$$

2. Einprozessorsystem \Rightarrow sequentielle Prozesse und nebenläufige Prozesse möglich

Aufgaben

- Erzeugen und Löschen von Prozessen (Fork)
- Prozessorzuteilung (Scheduling)
- Prozesskommunikation (Pipe, Sockets, Signale, ...)
- Synchronisation nebenläufiger Prozesse, die gemeinsame Daten benutzen

User-Mode und Kernel-Mode



Damit nebenläufige Prozesse sich nicht gegenseitig (zer-)stören, unterscheiden moderne Betriebssysteme zwei Zustände:

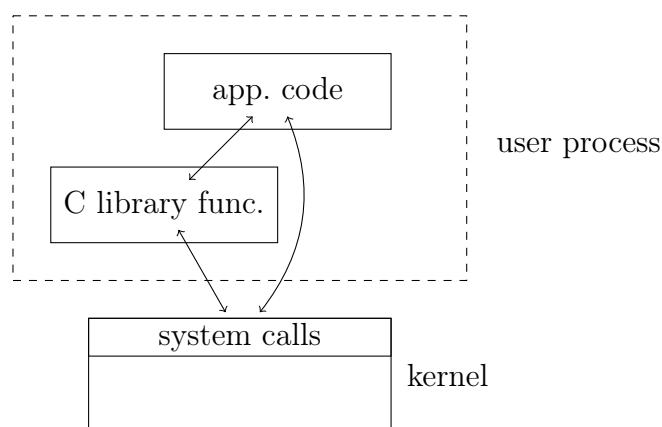
Eine Benutzeranwendung läuft i.a. im **unprivilegierten User-Mode**, der nur eingeschränkten Zugriff auf die Hardware erlaubt. Alle Betriebssystemkomponenten laufen im **privilegierten Kernel-Mode**, der uneingeschränkten Zugriff auf die Hardware erlaubt.

⇒ Spezielle Speicherbereiche dürfen nur im Kernel-Mode gelesen werden

Wie wird diese Unterscheidung realisiert?

Die Trennung von User- und Kernel-Mode benötigt Hardwareunterstützung im **Prozessorregister**:

1. Programmzähler: enthält die Adresse des aktuellen Befehls
2. Instruction Register: aktuell auszuführender Befehl
3. Program Status Word: Menge von Registern, die
4. Statusinformationen enthalten: User/Kernel Mode Bit, Interrupt enabled/disabled Bit, ...



1.2.2 Speicherverwaltung

- Zuteilung des verfügbaren physikalischen Speichers an Prozesse
- Einbeziehen des Hintergrundspeichers (Platte) durch **virtuelle Adressierung**

Techniken unter anderem **Swapping** (Ein-/Auslagern von Prozessen) und **Demand Paging**

2 Prozessverwaltung

2.1 Einführung

Ein **Prozess (process, task)** ist ein in Ausführung befindliches Programm bestehend aus

- einer Folge von Maschinenbefehlen, die durch das ausgeführte Programm festgelegt sind (program code, text section),
- dem Inhalt des Stapel-Speichers (stack), auf dem temporäre Variablen und Parameter für Funktionsaufrufe verwaltet werden
- dem Inhalt des Speichers, in dem die globale Daten des Prozesses gehalten werden (data section) und
- Verwaltungsinformationen, die den aktuellen Bearbeitungszustand (Kontext) beschreiben.

Bemerkung: Der aktuelle Bearbeitungszustand wird u.a. durch den Programmzähler und die Registerinhalte des Prozessors beschrieben (internal state).

Die Verwaltungsinformationen speichert das Betriebssystem in der **Prozesstabelle**, die pro Prozess einen Eintrag enthält. Die Einträge in der Prozesstabelle heißen **Prozesskontrollblock** oder auch **Prozesseitblock (process control block)** und beinhalten:

- Prozessidentifikationsnummer (pid)
- Prozesszustand
- Programmzähler
- Registerinhalte des Prozessors (CPU)
- Daten, die die Bearbeitungsfolge regeln (Prioritäten)
- Zeiger auf Text-, Daten-, Stacksegment
- Accounting-Daten zur Abrechnung von benutzten
- Betriebsmitteln (z.B. die bisher verbrauchte CPU-Zeit)
- Daten über Ein-/ Ausgabegeräte, die dem Prozess zugeordnet sind
- Daten über geöffnete Dateien und Netzwerkverbindungen
- Parameter, die die Zugriffsrechte des Prozesses beschreiben

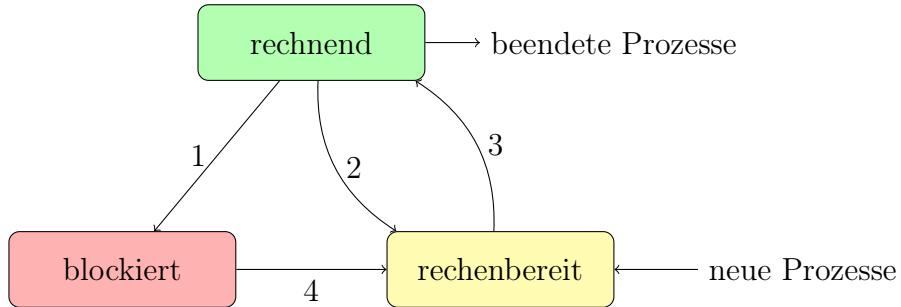
Die Prozesstabelle kann mit einer doppelt verketteten Liste realisiert werden. Außerdem wird zwischen dem **Hardwarekontext** (Registerinhalt, Befehlzzähler, ...) und **Softwarekontext** (Prozesszustand, Prozess-ID, ...) unterschieden.

Definition 8. Ein Prozess befindet sich genau in einem der folgenden Zustände:

rechnend - Der Prozessor ist dem Prozess zugeteilt

rechenbereit - Der Prozess ist ausführbar und wartet auf Prozessorzuteilung

blockiert - Der Prozess wartet auf ein Ereignis, z.B. auf Eingabe, Ausgabe, Semaphor



1. Prozess blockiert, weil er auf Eingabe wartet
2. Scheduler wählt einen anderen Prozess aus
3. Scheduler wählt diesen Prozess aus
4. Eingabe vorhanden

2.1.1 Abnormales Prozessverhalten

Definition 9. Ein Prozess befindet sich in einem

Deadlock - Der Prozess befindet sich im Zustand blockiert und es tritt kein Zustandswechsel mehr ein

Livelock - Der Prozess wechselt zwar zwischen den Zuständen, kommt aber in seiner Programmausführung nicht weiter

Starvation - Man sagt ein Prozess verhungert, falls er sich im Zustand rechenbereit befindet und es tritt kein Zustandswechsel ein.

2.1.2 Unterbrechungen

Unterbrechungen sind das Schlüsselkonzept zur effizienten Ablaufsteuerung.

1. Hardware Interrupts (asynchrone Unterbrechung)
 - I/O-Controller (Platte, Tastatur, Network Interface Card(NIC))
 - Timer: ermöglicht dem Betriebssystem periodisch bestimmte Arbeiten durchzuführen
2. Software Interrupts (auch: Traps, Program Interrupts) (synchrone Unterbrechung)
 - Systemdienstaufruf (System Call): `read`, `write`, `open`, `close`, ...

- Division durch Null
- ungültige Speicheradresse

Definition 10. Der enthält für jeden Unterbrechungstyp (Disk-Interrupt, Terminal-Interrupt, ...) die Adresse einer zugehörigen Unterbrechungsbehandlungsroutine.

Beispiel für eine asynchrone Unterbrechung:

Prozess *A* rechnet, Prozess *B* wartet auf Eingabe von der Platte.
 ⇒ Disk-Interrupt (Daten für Prozeß *B* sind eingelesen)

1. Programmzähler etc. von Prozeß *A* werden auf den Stack geschrieben (Hardware).
2. Unterbrechungsbehandlungsroutine wird geladen.
3. Unterbrechungsbehandlungsroutine schreibt die aktuellen Registerinhalte in den Prozeßleitblock, der Stack wird rekonstruiert (Prozeßzustand von *A* := rechenbereit) (Assembler).
4. C-Routine setzt Prozeß *B* auf rechenbereit.
5. Scheduler wird aktiv.

2.1.3 Erzeugen von Prozessen

- Der erste Prozess wird beim Systemstart vom Betriebssystem erzeugt
- Neue Prozesse können von einem existierenden Prozess durch spezielle Systemaufrufe erzeugt werden
- POSIX-Systeme benutzen hierzu den `fork()` Systemaufruf. Der neu erzeugte Prozess (child) ist eine exakte Kopie des erzeugenden Prozesses (parent), der **nebenläufig** zum erzeugenden Prozess läuft. Eltern und Kindprozess haben zudem einen getrennten Adressraum.
- Da beliebige Prozesse neue Prozesse erzeugen können, entsteht eine Hierarchie von Prozessen.

FORK(2)

Linux Programmer's Manual

FORK(2)

NAME

`fork - create a child process`

SYNOPSIS

```
#include <sys/types.h>
#include <unistd.h>

pid_t fork(void);
```

DESCRIPTION

`fork()` creates a new process by duplicating the calling process. The new process is referred to as the child process. The calling process is referred to as the parent process.

The child process and the parent process run in separate memory spaces. At the time of fork() both memory spaces have the same content. Memory writes, file mappings (`mmap(2)`), and unmappings (`munmap(2)`) performed by one of the processes do not affect the other.

The child process is an exact duplicate of the parent process except for the following points:

- * The child has its own unique process ID, and this PID does not match the ID of any existing process group (`setpgid(2)`) or session.
- * The child's parent process ID is the same as the parent's process ID.
- * The child does not inherit its parent's memory locks (`mlock(2)`, `mlockall(2)`).
- * Process resource utilizations (`getrusage(2)`) and CPU time counters (`times(2)`) are reset to zero in the child.
- * The child's set of pending signals is initially empty (`sigpending(2)`).
- * The child does not inherit semaphore adjustments from its parent (`semop(2)`).
- * The child does not inherit process-associated record locks from its parent (`fcntl(2)`). (On the other hand, it does inherit `fcntl(2)` open file description locks and `flock(2)` locks from its parent.)
- * The child does not inherit timers from its parent (`setitimer(2)`, `alarm(2)`, `timer_create(2)`).
- * The child does not inherit outstanding asynchronous I/O operations from its parent (`aio_read(3)`, `aio_write(3)`), nor does it inherit any asynchronous I/O contexts from its parent (see `io_setup(2)`).

RETURN VALUE

On success, the PID of the child process is returned in the parent, and 0 is returned in the child. On failure, -1 is returned in the parent, no child process is created, and `errno` is set appropriately.

...

Linux

2017-09-15

FORK(2)

2.1.4 Beenden von Prozessen

- Prozesse können in POSIX-Systemen mit Hilfe des `execve()` Systemaufrufs das ausgeführte Programm durch ein neues ersetzen.
- Prozesse können sich selbst durch den `exit()` Systemaufruf beenden und damit aus dem System entfernen.

- Ein Prozess beendet sich wenn ein Fehler festgestellt wird, der den weiteren Ablauf des Prozesses nicht möglich macht. Zum Beispiel der Aufruf `make main.c`, wenn die Datei `main.c` nicht existiert.
- Ein Prozess wird beendet, wenn vom Prozess selbst verursachte schwerwiegende Fehler vorliegen. Zum Beispiel beim Aufrufen unzulässiger Befehle, Division durch Null oder Zugriff auf ungültige Speicheradressen
- Ein Prozess kann die von ihm erzeugten Prozesse beenden, falls
 - ein Prozess nicht länger gebraucht wird oder falls
 - der erzeugende Prozess sich beendet und es nicht gewünscht/erlaubt ist, dass die erzeugten Prozesse unabhängig weiter existieren (**cascading termination**).
- Auf die Beendigung eines erzeugten Prozesses kann in POSIX-Systemen mit Hilfe des `wait()` Systemaufrufs gewartet werden.
- Der Ergebniswert (**return code**) eines sich beendenden Prozesses wird an den erzeugenden Prozess weitergeleitet.

2.2 Scheduling

Klassen von Schedulingverfahren

1. deterministisch ↔ nicht deterministisch

auch: Offline- versus Online-Verfahren

Beim deterministischen Scheduling ist der Rechenzeitbedarf einer Anwendung bzw. eine Schätzung für die Rechenzeit bekannt. Häufig werden auch alle Ankunftszeiten der Jobs im voraus als bekannt angenommen.

2. verdrängend ↔ nicht verdrängend (preemptive ↔ non-preemptive)

Bei **verdrängenden Verfahren** darf ein laufender Prozess unterbrochen werden (z.B. Timesharing, Prioritäten-Verfahren mit Unterbrechung). Bei **nicht-verdängenden Verfahren** bestehen asynchrone Unterbrechungen nur aus Device Interrupts. Der Scheduler lässt den ausgewählten Prozess so lange laufen, bis er blockiert (entweder wegen Ein-/Ausgabe oder weil er auf einen anderen Prozess wartet) oder freiwillig die CPU freigibt. Nach der Unterbrechungsbehandlung rechnet der unterbrochene Prozess weiter.

Ziele von Schedulingstrategien

Der Entwurf einer Schedulingstrategie hängt von mehreren Faktoren, wie der Umgebung (Stapelverarbeitung, Interaktivität, Echtzeit) ab. Einige Ziele sind jedoch auch für alle Systeme wünschenswert. Siehe Auflistung

Alle Systeme

- Fairness - jeder Prozess bekommt einen fairen Anteil der Rechenzeit
- Policy Enforcement - vorgegebene Strategien werden durchgesetzt
- Balance - alle Teile des Systems sind ausgelastet

Stapelverarbeitungssysteme

- Durchsatz – Maximieren der Jobs pro Stunde
- Durchlaufzeit – Minimieren der Zeit vom Start bis zur Beendigung
- CPU – die CPU ist immer ausgelastet

Interaktive Systeme

- Antwortzeit – schnelle Antwort auf Anfragen
- Proportionalität – Erwartungen des Benutzers erfüllen

Echtzeitsysteme

- Deadlines einhalten – Datenverlust vermeiden
- Vorhersagbarkeit – Qualitätseinbußen in Multimediasystemen vermeiden

Bezeichnungen

$W_i :=$ Startzeit – Ankunftszeit

Wartezeit von Prozess P_i

$t_i :=$ Rechenzeit von Prozess P_i

$V_i := W_i + t_i$

Verweilzeit(Laufzeit, Ausführungszeit) von Prozess P_i

Im Falle von Dialogbetrieb ist die **Reaktions-** bzw. **Antwortzeit** relevant:

Die Aufenthaltszeit eines Auftrags im System, bis zum Eintreffen des ersten Resultats an einer Ausgabeschnittstelle, heißt **Reaktionszeit**.

Gütekriterien für Scheduling-Verfahren

1. CPU-Auslastung

Optimierungsstrategien:

- Ausnutzen von I/O-Wartezeiten durch Multitasking
- Lastenausgleich bei Mehrprozessorsystem

2. \overline{W} mittlere Wartezeit, mit dem Ziel \overline{W} zu minimieren

3. im Dialogbetrieb **Reaktionszeit/Antwortzeit** minimieren

4. **Fairness-Prinzip:** Prozesse mit langer Rechenzeit sollen länger warten als kurze Prozesse, aber kein Prozess darf verhungern.

$$\Rightarrow \frac{1}{n} \sum_{i=1}^n \frac{W_i}{t_i} \text{ minimieren}$$

Echtzeitsysteme

Ziel: Einhalten von Fristen

r_i (ready time): frühester Zeitpunkt, zu dem der Prozessor an P_i zugeteilt werden darf.

t_i Schätzung für maximale Ausführungszeit

d_i (deadline) : Zeitpunkt, an dem die Ausführung von P_i beendet sein muss

s_i Startzeit von Prozess P_i

Zeitbedingungen

$$r_i \leq s_i \quad \wedge \quad s_i + t_i \leq d_i$$

Definition 11. Ein Vergabeplan für eine Prozeßmenge $P = P_1, P_2, \dots, P_n$ mit gegebenen r_i, t_i, d_i heißt gültig, falls sich keine Ausführungszeiten auf einem Prozessor überlappen und sämtliche Zeitbedingungen eingehalten werden.

Wie wählt man ein geeignetes Schedulingverfahren aus?

1. **Beschreibung der Betriebsart**

(Stapelverarbeitung, Dialogsystem, Echtzeitsystem, ...)

2. **Beschreibung der Arbeitslast**

(interaktiv, CPU-intensiv, I/O intensiv, periodische Datenströme, Verteilung der Ankunftszeiten, Verteilung der Rechenzeiten, ...) \Rightarrow Gütekriterium wird festgelegt

3. **Bewertung**

analytische Modellierung mittels Warteschlangentheorie bzw. Simulation

Probleme:

- Prozessprofile (interaktiv, CPU-intensiv) sind i.a. nicht im voraus bekannt
- Rechenzeiten t_i sind i.a. nicht im voraus bekannt,
- Schedulingverfahren verursachen Overhead!

\Rightarrow Einsatz von einfachen **Heuristiken**, die die optimale Lösung annähern sollen

2.2.1 First-In-First-Out (FIFO)

auch FCFS (First Come First Served)

Eigenschaften: non-preemptive, einfache Verwaltung, fair



8	4	4	4
A	B	C	D

Abbildung 2: Originalreihenfolge

4	4	4	8
B	C	D	A

Abbildung 3: SJF Reihenfolge

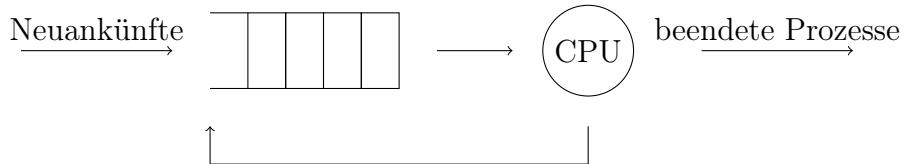
2.2.2 Shortest Job First (SJF)

- nicht unterbrechende Strategie
- nur optimal wenn alle Prozesse gleichzeitig eintreffen

Theorem 1. Bei gegebener Prozessmenge ist SJF optimal bzgl. \bar{W}

Approximation für SJF: Zeitscheibenverfahren

Jedem Prozeß wird ein Zeitquantum (sog. **Zeitscheibe**) an Rechenzeit zugeteilt. Die rechenbereiten Prozesse werden in einer FIFO-Warteschlange verwaltet. Nach Ablauf der Zeitscheibe werden nicht-beendete Prozesse wieder hinten in die Warteschlange einsortiert (**Round-Robin**).



Die Wartezeit eines Prozesses hängt ab:

- Anzahl Prozesse im System,
- Rechenzeitanforderung des Prozesses.

Vorteil: Zeitscheiben reduzieren die Antwortzeit: Sind bei Ankunft des Prozesses n Prozesse im System, bekommt der Prozeß nach maximal n Zeitscheiben die CPU zugeteilt

Nachteil: Zusätzlicher Aufwand für den Prozesswechsel

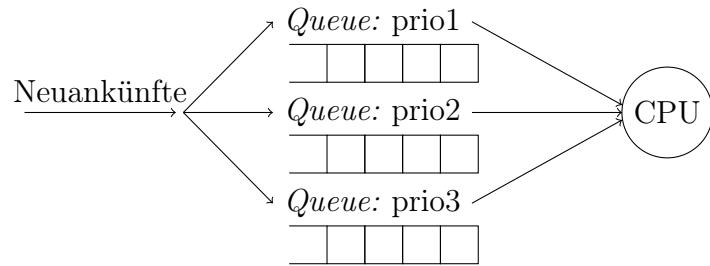
2.2.3 Prioritäten-basiertes Scheduling

verdrängend / nicht verdrängend

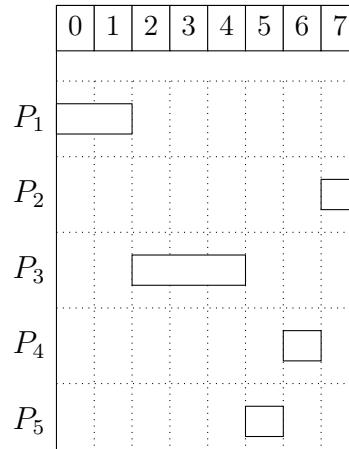
Verfahren:

Jedem Prozeß wird eine Priorität zugeordnet. Prozesse einer Priorität werden in derselben Warteschlange verwaltet. Innerhalb der Warteschlange wird FIFO angewandt. Die CPU wird dem ersten Prozeß in der nichtleeren Warteschlange höchster Priorität zugeordnet.

Bespiel:



	t	prio	Ankunftszeit
P_1	2	1	0
P_2	1	0	1
P_3	3	2	2
P_4	1	1	3
P_5	1	2	4



2.2.4 Betriebssystemkomponenten

Definition 12. Die **Scheduler-Komponente** implementiert die Schedulingstrategie des Systems und bestimmt, welcher Prozeß als nächster rechnen darf.

Die **Dispatcher-Komponente** ist für den Prozeßwechsel zuständig: Der Zustand des bisher laufenden Prozesses muss gesichert werden und der neu ausgewählte Prozeß muss rechenbereit gesetzt werden

2.2.5 Beispiel: Ablaufsteuerung unter UNIX

hier *BSD 4.4*

Prozesse haben eine Priorität zwischen 0 (**hoch**) und 127 (**niedrig**). Jedem Prozess sind zwei Prioritäten zugeordnet, eine im User-Mode und eine im Kernel-Mode:

1. Benutzerprozesse laufen i.d.R. im User-Mode mit Basispriorität PUSER = 50. Die Priorität im User-Mode wird im Feld `p_usrpri` im Prozeßleitblock gemerkt. Der Wert liegt im Bereich 50-127.
2. Bei einem Systemaufruf wechselt der Prozeß in den Kernel-Mode und wird i.d.R. blockiert, weil er auf ein Ereignis wartet (I/O, `sem_wait()`, `waitpid()`, ...). Im `p_priority` Feld im Prozeßleitblock wird die dem Ereignis zugeordnete Priorität gemerkt.
3. Tritt das Ereignis ein, wird der Prozeß in die zu `p_priority` gehörende Warteschlange einsortiert (**Priority Boosting**). Seine Basispriorität ändert sich dabei nicht! Nach Ablauf der Zeitscheibe wird er wieder gemäss der Basispriorität eingereiht.

2.3 Beispiele

2.3.1 4.3 BSD UNIX Scheduler

- Der Scheduler des 4.3 BSD UNIX Betriebssystems benutzt eine dem **Round-Robin Verfahren überlagerte Prioritätensteuerung**. Bei jedem Prozeßwechsel sucht der Dispatcher die nicht-leere Run-Queue mit höchster Priorität und startet den am längsten in ihr stehenden Prozeß.
- Die **Zeitscheibenlänge**, die ein Prozeß zugewiesen bekommt, beträgt eine Zehntelsekunde.
- Prioritäten werden zwischen 0 und 127 vergeben, wobei ein kleinerer Wert einer höheren Priorität entspricht. Die Basispriorität PUSER für Nutzerprozesse hat den Wert 50.
- Die Variable `p_usrpri` enthält die Prozeßpriorität, die vom Dispatcher ausgewertet wird.

Runqueue-Verwaltung

Um den Prozeß mit der höchsten Priorität **schnell** identifizieren zu können, benutzt der Dispatcher 32 Run-Queues, in die die Prozesse gemäß ihrer Priorität eingeordnet werden.

Die Nummer der Run-Queue wird gemäß

$$rq := \left[\frac{p_{\text{usrpri}}}{4} \right]$$

bestimmt.

Dynamische Prioritäten

Grundidee ist, daß CPU-intensive Prozesse an Priorität verlieren und kurze oder E/A-intensive Prozesse Priorität gewinnen.

Die Priorität eines Prozesses wird verändert, falls

1. der Prozess länger als eine Sekunde blockiert auf I/O gewartet hat: Priority Boosting
2. der Prozeß rechnet. Dann verliert der Prozeß an Priorität.
3. Zusätzlich wendet der BSD Scheduler einmal pro Sekunde einen **Vergissfilter** auf die Prioritäten an.

Vergissfilter

Damit die Prioritäten nicht ständig wachsen, wird der Wert `p_cpu` von **allen** Prozessen **einmal pro Sekunde** durch einen Vergissfilter korrigiert:

$$p_{\text{cpu}} := \frac{2 \cdot \text{load}}{2 \cdot \text{load} + 1} \cdot p_{\text{cpu}} + p_{\text{nice}}$$

Dabei bezeichnet `load` die durchschnittliche Anzahl der rechenbereiten Prozesse (inklusive des rechnenden Prozesses) über ein vorangegangenes 1-Minuten-Intervall.

Bemerkung:

1. Das BSD Schedulingverfahren ist responsiv dank Priority Boosting.
2. Prozesse mit geringer Priorität können verhungern (nicht starvation-free).
3. Verfahren skaliert nicht:
 - (a) Anpassen der Priorität aller Prozesse alle 4 Time Ticks
 - (b) Anpassen von `p_cpu` aller Prozesse einmal pro Sekunde gemäß dem Vergissfilter.

2.3.2 Linux

- Linux verwaltet und plant Threads.
- Diese werden als **Tasks** bezeichnet und mittels der Datenstruktur `task_struct` verwaltet. Jeder Thread besitzt eine eigene Task-Datenstruktur.
Beispiel: Ein Prozess mit 1 Thread wird mittels einer Task-Datenstruktur verwaltet. Ein Prozess mit n Threads wird mittels n Task-Datenstrukturen verwaltet.
- Um POSIX-konform zu bleiben, besitzen alle Tasks einer Anwendung (d.h. alle Threads eines Prozesses) dieselbe Prozess-ID (PID). **Beispiel:** Das Kommando `$ kill <pid>` terminiert alle Threads mit der angegebenen PID.
- Timesharing-Prozesse bekommen dynamische Prioritäten zwischen 100-139.
- Linux benutzt **statische** Prioritäten für sogenannte Echtzeitprozesse. Diese haben die höchsten Prioritäten 0-99. Es rechnet jeweils der rechenbereite Thread mit der höchsten Priorität.
 - **Real-Time FIFO:** Non-preemptive Scheduling-Verfahren. Ausnahme: Falls ein Thread mit höherer Priorität als der laufende rechenbereit wird, wird der laufende Thread unterbrochen und der Thread höherer Priorität darf rechnen.
 - **Real-Time Round Robin:** Zeitscheibenverfahren Auch hier läuft jeweils der Thread höchster Priorität. Aber bei mehreren Threads gleicher Priorität wird ein Zeitscheibenverfahren mit festem Quantum angewandt.

Achtung: Statische Prioritäten sind nur eine Voraussetzung zur Implementierung von Echtzeit-systemen, aber nicht hinreichend!

Linux 2.6.23 Completely Fair Scheduler (CFS) von Ingo Molnar (04/2007).

Kennzeichen:

1. Neue Datenstruktur: Keine verketteten Listen mehr, sondern Red-Black-Tree.
2. Task Group Scheduling:
Tasks einer Gruppe werden gemeinsam geschedult. Damit wird Fairness erreicht, wenn eine Task viele weitere Tasks erzeugt.

Angebliche Motivation: Bessere Performance auf Desktops, d.h. bessere Unterstützung von interaktiven Prozessen.

2.3.3 Red-Black-Tree (`rbtree.h`)

Allgemein

- Selbstbalancierter binärer Suchbaum mit garantierter
- Worst-Case-Komplexität $O(\log(n))$ (Rudolf Bayer, 1972)
- Tasks sortiert durch Depth-First-Suche von links nach rechts meist Zugriff auf linkes Element (cached)

Eigenschaften

Ein binärer Suchbaum heisst Red-Black-Tree, wenn folgende Eigenschaften erfüllt sind:

1. Jeder Knoten ist entweder rot oder schwarz.
2. Die Wurzel ist schwarz.
3. Jedes (externe) Blatt, d. h. `null`, ist schwarz.
4. Ein roter Knoten hat nur schwarze Kinder.
5. Für jeden Knoten enthalten alle Pfade, die an diesem Knoten starten und in einem Blatt des Teilbaumes dieses Knotens enden, die gleiche Anzahl schwarzer Knoten.

2.3.4 Funktionsweise CFS - Überblick

- Scheduling Entity (SE) kann Task oder Task-Group sein
- Red-Black-Tree ist nach `vruntime` geordnet: die bisher verbrauchte, gewichtete CPU-Zeit
⇒ Starvation vermeiden
- wenn `current` nicht mehr left-most Node ist (und Granularity-Zeit (Zeitscheibe) überschritten), wird `current` verdrängt.
- CFS versucht den Idealfall (?) zu erreichen, dass alle SEs gleiche `vruntime` haben
- `vruntime` wird auf `min_vruntime` gesetzt, wenn Task in die Runqueue (RB-Tree) kommt.
- Einfügen in Red-Black-Tree: $O(\log(n))$
Ermitteln des nächsten zu rechnenden Threads: Zeiger auf das Element ganz links: $O(1)$

Einfluss der Priorität:

- vruntime wird periodisch um `Laufzeit/nr_running` gewichtet nach prio erhöht:
Zeit vergeht für Prozesse geringerer Priorität schneller! \Rightarrow fließt in die Neuberechnung der vruntime ein.
Mauerer: *CFS doesn't use priorities directly but instead uses them as a decay factor for the time a task is permitted to execute. ...*
This means that the time a task is permitted to execute dissipates more quickly for a lower-priority task than for a higher-priority task. That's an elegant solution to avoid maintaining run queues per priority. \Rightarrow keine festen Zeitscheiben: 10 ms - 5120 ms
- nice-Kommando

Bemerkung

1. Analog zum BSD-Scheduling macht Linux prioritäten-basiertes Scheduling.
2. Es wird nur eine Warteschlange benutzt, die als Red-Black-Tree organisiert ist.
3. Der Red-Black-Tree ist nach der berechneten `vruntime` sortiert.
4. In die Berechnung der `vruntime` geht die vom Elternprozeß geerbte Priorität, der nice-Wert und - analog zu BSD - die bisher verbrauchte CPU-Zeit ein.
5. Einfügen in Red-Black-Tree: $O(\log(n))$ Ermitteln des nächsten zu rechnenden Threads: $O(1)$
6. Die `vruntime` eines Prozesses wird nur neu berechnet, nachdem er gerechnet hat.
7. Kein explizites Priority Boosting.
8. Da die bisher verbrauchte CPU-Zeit in die Berechnung der `vruntime` mit eingeht, verschlechtern sich Prozesse, die rechnen. \Rightarrow Starvation free.

2.3.5 Echtzeit-Scheduling unter Linux

Seit dem Linux-Kernel 3.14 steht das sogenannte Deadline-Scheduling zur Verfügung:

- Kombination zweier Scheduling-Verfahren:
Earliest-Deadline-First-Scheduling (EDF) sowie Constant-Bandwidth-Server (CBS)
- Der Deadline-Scheduler berücksichtigt bei der Planung für jede Task 3 Parameter:
 1. `runtime`: maximale CPU-Rechenzeit pro Zeitperiode
 2. `deadline`: Zeitpunkt, an dem der Task spätestens fertig gerechnet haben muß
 3. `period`: Zeitperiode während der der Task die definierte `runtime` bekommt. Dies kann z.B. der Kehrwert der maximalen Frequenz des Auftretens der Task sein.
 4. Überschreitet eine Task ihre spezifizierte `runtime`, so wird sie unterbrochen und darf erst nach der `period` weiterrechnen.
 5. Ko-Existenz von Realtime- und Timesharing-Tasks: Timesharing-Tasks haben eine geringere Priorität.
 \Rightarrow Gefahr von Starvation!
- Der Deadline-Scheduler wählt diejenige Task als nächste zu rechnende Task aus, deren Deadline am nächsten ist (gemäß EDF).

2.4 Thread-Systeme

Definition 13. Ein **Leichtgewichtprozeß (Thread)** ist eine Ausführungseinheit mit minimalen Zustandsinformationen (z.B. Programmzähler, Registerinhalte, Thread-Priorität).

Definition 14. Eine **Task** ist eine Verwaltungseinheit, in der die Systemressourcen wie z.B. Hauptspeicher, I/O-Kanäle etc., die einer oder mehreren Threads zugeteilt sind, zusammengefaßt sind.

2.4.1 Kernel-Level-Threads

Beispiele: Mach, Chorus, ...

SunOS 5.x (Solaris 2.x), Linux, Windows NT, ...

2.4.2 User-Level-Threads

Threads auf Anwendungsebene

- Threadbibliotheken als Ergänzung zu gängigen Programmiersprachen.
Beispiel: POSIX-Threads, SunOS 4.1x LWP, ...
- (Parallele) Sprachen mit neuer Laufzeitumgebung.
Beispiel: Cedar, Mesa, Java, OpenMP, ...

Beispiel: Solaris: 2-stufiges System

- Kernel unterstützt LWPs (Lightweight Processes).
- Es werden zwei Threadbibliotheken (**POSIX libpthread**, **Solaris libthread**) angeboten.

Beispiel: POSIX (Portable Operating Systems based on UNIX), libpthread-Spezifikation von Funktionen für:

- Thread-Verwaltung (Erzeugen, Löschen, ...),
- Synchronisation (Mutex-, Bedingungsvariable),
- Scheduling (prioritätengesteuertes FIFO, zeitscheibengesteuertes Round Robin), Signalbehandlung, ...

2.4.3 Thread Safety

Problem: Was passiert, wenn mehrere Threads nebenläufig dieselbe Funktion aufrufen?

Definition 15. Eine Funktion heißt **thread-safe** bzw. **reentrant**, wenn sie das korrekte Ergebnis liefert, auch wenn sie von mehreren Threads nebenläufig aufgerufen wird.

2.4.4 Parallelisierungskonzepte

- **Master-Worker-Modell:** Arbeitspakete werden vom sogenannten Master-Thread an die Worker-Threads verteilt.
- **Pool-of-Thread:** Eine vorkonfigurierte Anzahl Threads wird erzeugt. Die Threads holen sich dynamisch nach Fertigstellung eines Jobs die nächste Arbeit ab. In der Regel wird dies über eine gemeinsame Datenstruktur, die die Beschreibung der Aufträge/Jobs enthält, die sogenannte **Work-Queue**, gesteuert. Dieser Ansatz kann mit dem Master-Worker-Modell kombiniert werden.
- **Pipeline:** Threads arbeiten am Fließband unterschiedliche Teilaufgaben an den Eingabedaten ab.
Beispiel: Intrusion Detection Systeme: Eintreffende Pakete werden nacheinander von verschiedenen Threads auf verschiedene Angriffssignaturen geprüft.

2.4.5 Beispiel: Java-Threads

Klasse `Thread` mit Methode `run`

Durch Vererbung können neue Klassen definiert werden. Die neue Klasse

- erbt die Methoden der Basisklasse,
- kann bestehende Methoden überschreiben,
- neue Methoden hinzufügen.

Zugriff auf Elemente und Methoden eines Objekts:

- **public:** Zugriff sowohl von außen als auch von allen Nachfolgern aus erlaubt.
- **private:** Zugriff nur innerhalb der Klasse möglich.
- ...

2.5 Kommunikation

Verfahren zur *Interprozesskommunikation (IPC)*

2.5.1 IPC - Signale

Ziel:

schnelle Reaktion auf bestimmte vordefinierte Ereignisse durch **Schicken eines Signals**

Hinweis:

Nicht zum Datenaustausch geeignet.

Anwendung:

- Prozesskontrolle (Job control), z.B. Beenden eines (fehlerhaften) Programms (`Ctrl-c`)

Eigenschaften:

- Signale unterbrechen die aktuelle Arbeit des Prozesses und erlauben daher eine extrem schnelle Reaktion auf ein Ereignis.
- Signale erlauben es nicht, direkt Daten auszutauschen.
- Signale sind auf das lokale System begrenzt.
- Prozesse mit gleicher effektiver UserID (wird vom Vaterprozess geerbt) können sich Signale schicken.
Ausnahme: Root-Prozesse können Signale an alle Prozesse schicken.
- Signale werden gemerkt, d.h. ein Signal erreicht auch einen schlafenden/blockierten Prozeß nach dem Aufwachen.

Lösung: kill pid

Empfängerprozess hat keine CPU-Zeit, wenn das Signal gesendet wird.

Wie erhält der Empfängerprozess das Signal?

Realisierung:

- Signale werden vom Betriebssystem als asynchrone Softwareunterbrechungen realisiert.
- Bit-Map für Signale im Prozeßleitblock
⇒ Für ein gesendetes Signal wird das entsprechende Bit in der Bit-Map im Prozeßleitblock des Empfängers gesetzt.
- Jedem Signal ist eine Signalbehandlungsroutine action zugeordnet:
 1. **default action**
 2. benutzerdefinierte Signal-Behandlungs Routinen sind möglich: Benutzer kann einen **signal handler** installieren. Dieser fängt das Signal ab (**catching a signal**) und ruft die vom Benutzer spezifizierte Behandlungsroutine auf.
 3. Signal kann ignoriert werden

Achtung: Achtung: Nicht alle Signale können abgefangen bzw. ignoriert werden. Ausnahmen: SIGKILL, SIGSTOP

Beispiele für in 4.4 BSD definierte Signale

Name	Default Action	Description
SIGKILL	terminate process	kill program
SIGINT	terminate process	interrupt program
SIGHUP	terminate process	terminal line hangup
SIGTSTP	stop process	stop signal generated from keyboard
SIGQUIT	create core image	quit program
SIGBUS	create core image	bus error
SIGFPE	create core image	floating-point exception
SIGSYS	create core image	non-existent system call invoked
SIGCONT	discard signal	continue after stop
SIGCHLD	discard signal	child status has changed

2.5.2 IPC - Pipes

- Unidirektonaler Datenfluß von einem Daten erzeugenden Prozeß zu einem Daten verarbeitenden Prozeß.
- Daten werden gepuffert in FIFO-Reihenfolge zugestellt.
- Keine Unterstützung zur Strukturierung von Daten. (Empfänger muss den Bytestrom oftmals wieder mit einem Scanner/Parser in Datenblöcke zerlegen.)
- Pipes werden beispielsweise zwischen Vater/Kind-Prozessen eingesetzt.
- Sogenannte named pipes erlauben beliebigen Prozessen an einer Pipe teilzunehmen.
- Pipes sind auf das lokale System beschränkt

Beispiel: Verwaltung von I/O-Kanälen unter UNIX

Ein I/O - Kanal wird durch einen sogenannten **Descriptor** beschrieben.

Ein Descriptor wird als Integerwert realisiert und ist der Index für den Tabelleneintrag in der sogenannten **Descriptor-Tabelle**.

Die Descriptor-Tabelle enthält für jeden geöffneten I/O-Kanal einen Eintrag:

Datei: File-Descriptor: Rückgabewert vom `open()`
Socket: Socket-Descriptor: Rückgabewert vom `socket()` Die Descriptor-Tabelle wird im Pipe: 2 Pipe-Descriptoren: im `pipe()` gesetzt
Prozeßleitblock gespeichert. Die **Tabelleneinträge** der Descriptor-Tabelle bestehen aus:

- **Datei:**
Verweis auf *inode*, aktuelle Position in der Datei, ...
- **Pipe:**
aktuelle Position für Lesen bzw. Schreiben
- **Socket:**
Speicherplatzadresse für Daten, Verwaltungsinfos wie z.B. benutztes Transportprotokoll (UDP, TCP), ...

Lösung: `int pipe(int filedes[2])`

PIPE(2)

Linux Programmer's Manual

PIPE(2)

NAME

`pipe` - create pipe

SYNOPSIS

```
#include <unistd.h>
```

```
int pipe(int pipefd[2]);
```

DESCRIPTION

pipe() creates a pipe, a unidirectional data channel that can be used for interprocess communication. The array pipefd is used to return two file descriptors referring to the ends of the pipe. pipefd[0] refers to the read end of the pipe. pipefd[1] refers to the write end of the pipe. Data written to the write end of the pipe is buffered by the kernel until it is read from the read end of the pipe. For further details, see pipe(7).

...

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and errno is set appropriately.

On Linux (and other systems), pipe() does not modify pipefd on failure. A requirement standardizing this behavior was added in POSIX.1-2016. The Linux-specific pipe2() system call likewise does not modify pipefd on failure.

...

EXAMPLE

The following program creates a pipe, and then fork(2)s to create a child process; the child inherits a duplicate set of file descriptors that refer to the same pipe. After the fork(2), each process closes the file descriptors that it doesn't need for the pipe (see pipe(7)). The parent then writes the string contained in the program's command-line argument to the pipe, and the child reads this string a byte at a time from the pipe and echoes it on standard output.

Program source

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

int
main(int argc, char *argv[])
{
    int pipefd[2];
    pid_t cpid;
    char buf;

    if (argc != 2) {
        fprintf(stderr, "Usage: %s <string>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    if (pipe(pipefd) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);
    }

    cpid = fork();
    if (cpid == -1) {
        perror("fork");
```

```

        exit(EXIT_FAILURE);
    }

    if (cpid == 0) { /* Child reads from pipe */
        close(pipefd[1]); /* Close unused write end */

        while (read(pipefd[0], &buf, 1) > 0)
            write(STDOUT_FILENO, &buf, 1);

        write(STDOUT_FILENO, "\n", 1);
        close(pipefd[0]);
        _exit(EXIT_SUCCESS);

    } else { /* Parent writes argv[1] to pipe */
        close(pipefd[0]); /* Close unused read end */
        write(pipefd[1], argv[1], strlen(argv[1]));
        close(pipefd[1]); /* Reader will see EOF */
        wait(NULL); /* Wait for child */
        exit(EXIT_SUCCESS);
    }
}

```

Linux

2017-11-26

PIPE(2)

2.5.3 IPC - Gemeinsamer Speicher (Shared Memory)

- Mehrere Prozesse teilen sich einen gemeinsamen Speicherbereich.
- Schnelle Kommunikation, da gemeinsame Daten direkt gelesen werden können.
- Programmieraufwand zur korrekten Synchronisation der Prozesse erforderlich.
- Verwaltung dynamischer, komplexer Datenstrukturen erzwingt häufig durch die erforderliche Synchronisation eine Serialisierung der beteiligten Prozesse.
- Gemeinsamer Speicher ist auf das lokale System begrenzt. Es existieren Vorschläge zur verteilten Realisierung von gemeinsamem Speicher (Distributed Shared Memory), wobei aber der Vorteil (schnelle Kommunikation) stark beeinträchtigt wird.

2.5.4 IPC - Nachrichten: Sockets

- Allgemeine Schnittstelle zur Interprozeßkommunikation, die **nicht auf lokale Prozesse beschränkt** ist.
- Entwickelt und zuerst implementiert im 4.2BSD UNIX (1984). Mittlerweile auf allen wichtigen Betriebssystemen verfügbar.
- In vielen Betriebssystemen als Teil des Betriebssystemkerns implementiert.
- Sockets realisieren einen abstrakten, protokollunabhängigen Zugriff auf die Dienste, die in der Regel von einem **Transportprotokoll** bereitgestellt werden.
- Gute Portabilität des Quellcodes, selbst über verschiedene Plattformen.

Ziele und Eigenschaften:

- **Transparenz:** Die Kommunikation soll davon unabhängig sein, ob sich die Kommunikationspartner auf einem lokalen System befinden oder räumlich getrennt sind.
- Die Endpunkte eines Kommunikationskanals müssen durch ein einheitliches Objekt dargestellt werden. ⇒ **socket**
- Die Schnittstelle muss unabhängig von konkreten Protokollen für lokale Kommunikation bzw. Netzkommunikation sein. ⇒ Man unterscheidet Protokollfamilien (Communication Domains).
Beispiel: Internet Domain, UNIX Domain
- Prozesse müssen in der Lage sein, andere Kommunikationsendpunkte zu lokalisieren. ⇒ Socket-Adressen (spezifisch für die einzelnen Domänen)

2.5.5 Übersicht über die Socket-Schnittstelle

- **bind(socket, address, length):**
Bindet eine Adresse an einen lokalen Endpunkt.
- **connect(socket, address, length):**
Herstellen einer Verbindung zu einem zweiten Socket, der durch die Adresse bestimmt ist.
- **listen(socket, backlog):**
Anzeigen, daß eingehende Verbindungswünsche akzeptiert werden.
- **accept(socket, address, length):**
Annahme einer Verbindung.
- **write(...), send(...), sendto(...):**
Übertragen einer Bytefolge über einen Socket.
- **read(...), recv(...), recvfrom(...):**
Empfangen einer Bytefolge von einem Socket.
- **close(socket):**
Terminieren einer Verbindung und Freigabe des Kommunikationsendpunkts.
- **shutdown(socket, how):** (Teilweise) Beendigung einer Verbindung.
- **setsockopt(), getsockopt():** Setzen und Lesen von Socket-Optionen.
- **getsockname(), getpeername():** Erfragen der lokalen bzw. entfernten Adresse.
- Eine Reihe von Hilfsfunktionen zur Umwandlung von Namen in Adressen und zur Konvertierung von Zahlen in eine einheitliche Byteordnung, die sogenannte **Network Byte Order**.
Eindeutige Darstellung von (unsigned) 16- bzw. 32-Bit Integer (short/long):
htons(), **htonl()**: host to network order
ntohs(), **ntohl()**: network to host order

Socket-Aufrufe für verbindungsloses Protokoll

- Beim verbindungslosen Protokoll werden auf beiden Seiten Sockets geöffnet und an eine Adresse gebunden.
- Der `recvfrom()`-Aufruf **blockiert** den Prozeß, bis eine Nachricht eingetroffen ist. Die Adresse des sendenden Prozesses wird dem empfangenden Prozeß mitgeteilt.
- Der `sendto()`-Aufruf sendet die Daten an die angegebene Adresse.
- Beim Schließen eines Endpunkts ist keine Interaktion erforderlich.

Prozess A	Prozess B
<code>socket()</code>	<code>socket()</code>
<code>bind()</code>	<code>bind()</code>
<code>recvfrom()</code>	<code>sendto()</code>
<code>close()</code>	<code>close()</code>

Socket-Aufrufe für verbindungsorientiertes Protokoll

- Beim verbindungsorientierten Protokoll wird zunächst von einer Seite ein Socket geöffnet, über den Verbindungswünsche entgegen genommen werden.
- Der `listen()`-Aufruf markiert das Socket als passives Socket, d.h. es wird nur dazu benutzt ankommende Nachrichten anzunehmen.
- Der `accept()`-Aufruf **blockiert** den Prozeß, bis eine Verbindung etabliert ist und liefert einen neuen Socket für diese Verbindung.
- Die `read()` und `write()` Aufrufe sind **blockierend**.
- Nach der Auflösung der Verbindung kann mit einem erneuten Aufruf von `accept()` eine weitere Verbindung entgegen genommen werden.

SOCKET(2)

Linux Programmer's Manual

SOCKET(2)

NAME

`socket` - create an endpoint for communication

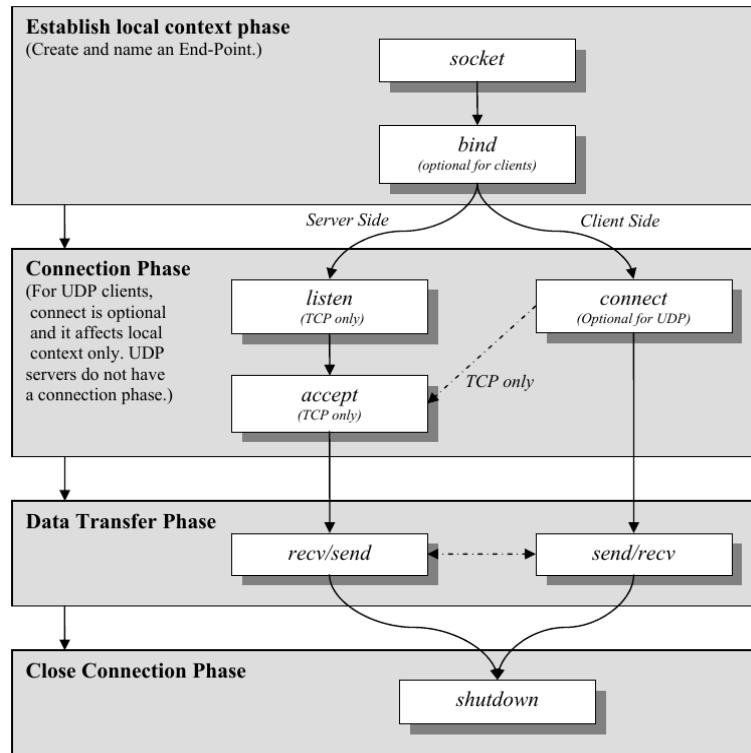
SYNOPSIS

```
#include <sys/types.h>          /* See NOTES */
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

DESCRIPTION

`socket()` creates an endpoint for communication and returns a file descriptor that refers to that endpoint. The file descriptor returned by a successful call will be the lowest-numbered file descriptor not currently open for the process.



Ablauf der Socket-Kommunikation

The domain argument specifies a communication domain; this selects the protocol family which will be used for communication. These families are defined in `<sys/socket.h>`. The currently understood formats include:

Name	Purpose	Man page
AF_UNIX, AF_LOCAL	Local communication	unix(7)
AF_INET	IPv4 Internet protocols	ip(7)
AF_INET6	IPv6 Internet protocols	ipv6(7)
AF_IPX	IPX - Novell protocols	
AF_NETLINK	Kernel user interface device	netlink(7)
AF_X25	ITU-T X.25 / ISO-8208 protocol	x25(7)
AF_AX25	Amateur radio AX.25 protocol	
AF_ATMPVC	Access to raw ATM PVCs	
AF_APPLETALK	AppleTalk	ddp(7)
AF_PACKET	Low level packet interface	packet(7)
AF_ALG	Interface to kernel crypto API	

The socket has the indicated type, which specifies the communication semantics. Currently defined types are:

SOCK_STREAM	Provides sequenced, reliable, two-way, connection-based byte streams. An out-of-band data transmission mechanism may be supported.
SOCK_DGRAM	Supports datagrams (connectionless, unreliable messages of a fixed maximum length).
SOCK_SEQPACKET	Provides a sequenced, reliable, two-way connection-

based data transmission path for datagrams of fixed maximum length; a consumer is required to read an entire packet with each input system call.

SOCK_RAW	Provides raw network protocol access.
SOCK_RDM	Provides a reliable datagram layer that does not guarantee ordering.
SOCK_PACKET	Obsolete and should not be used in new programs; see packet(7).

Some socket types may not be implemented by all protocol families.

Since Linux 2.6.27, the type argument serves a second purpose: in addition to specifying a socket type, it may include the bitwise OR of any of the following values, to modify the behavior of socket():

SOCK_NONBLOCK	Set the O_NONBLOCK file status flag on the new open file description. Using this flag saves extra calls to fcntl(2) to achieve the same result.
SOCK_CLOEXEC	Set the close-on-exec (FD_CLOEXEC) flag on the new file descriptor. See the description of the O_CLOEXEC flag in open(2) for reasons why this may be useful.

The protocol specifies a particular protocol to be used with the socket. Normally only a single protocol exists to support a particular socket type within a given protocol family, in which case protocol can be specified as 0. However, it is possible that many protocols may exist, in which case a particular protocol must be specified in this manner. The protocol number to use is specific to the “communication domain” in which communication is to take place; see protocols(5). See getprotoent(3) on how to map protocol name strings to protocol numbers.

Sockets of type SOCK_STREAM are full-duplex byte streams. They do not preserve record boundaries. A stream socket must be in a connected state before any data may be sent or received on it. A connection to another socket is created with a connect(2) call. Once connected, data may be transferred using read(2) and write(2) calls or some variant of the send(2) and recv(2) calls. When a session has been completed a close(2) may be performed. Out-of-band data may also be transmitted as described in send(2) and received as described in recv(2).

The communications protocols which implement a SOCK_STREAM ensure that data is not lost or duplicated. If a piece of data for which the peer protocol has buffer space cannot be successfully transmitted within a reasonable length of time, then the connection is considered to be dead. When SO_KEEPALIVE is enabled on the socket the protocol checks in a protocol-specific manner if the other end is still alive. A SIGPIPE signal is raised if a process sends or receives on a broken stream; this causes naive processes, which do not handle the signal, to exit. SOCK_SEQPACKET sockets employ the same system calls as SOCK_STREAM sockets. The only difference is that read(2) calls will return only the amount of data requested, and any data remaining in the arriving packet will be discarded. Also all message boundaries in incoming datagrams are preserved.

`SOCK_DGRAM` and `SOCK_RAW` sockets allow sending of datagrams to correspondents named in `sendto(2)` calls. Datagrams are generally received with `recvfrom(2)`, which returns the next datagram along with the address of its sender.

`SOCK_PACKET` is an obsolete socket type to receive raw packets directly from the device driver. Use `packet(7)` instead.

An `fcntl(2)` `F_SETOWN` operation can be used to specify a process or process group to receive a `SIGURG` signal when the out-of-band data arrives or `SIGPIPE` signal when a `SOCK_STREAM` connection breaks unexpectedly. This operation may also be used to set the process or process group that receives the I/O and asynchronous notification of I/O events via `SIGIO`. Using `F_SETOWN` is equivalent to an `ioctl(2)` call with the `FIOSETOWN` or `SIOCSPGRP` argument.

When the network signals an error condition to the protocol module (e.g., using an ICMP message for IP) the pending error flag is set for the socket. The next operation on this socket will return the error code of the pending error. For some protocols it is possible to enable a per-socket error queue to retrieve detailed information about the error; see `IP_RECVERR` in `ip(7)`.

The operation of sockets is controlled by socket level options. These options are defined in `<sys/socket.h>`. The functions `setsockopt(2)` and `getsockopt(2)` are used to set and get options, respectively.

RETURN VALUE

On success, a file descriptor for the new socket is returned. On error, -1 is returned, and `errno` is set appropriately.

ERRORS

`EACCES` Permission to create a socket of the specified type and/or protocol is denied.

`EAFNOSUPPORT`

The implementation does not support the specified address family.

`EINVAL` Unknown protocol, or protocol family not available.

`EINVAL` Invalid flags in type.

`EMFILE` The per-process limit on the number of open file descriptors has been reached.

`ENFILE` The system-wide limit on the total number of open files has been reached.

`ENOBUFS` or `ENOMEM`

Insufficient memory is available. The socket cannot be created until sufficient resources are freed.

`EPROTONOSUPPORT`

The protocol type or the specified protocol is not supported within this domain.

Other errors may be generated by the underlying protocol modules.

...

Linux

2017-09-15

SOCKET(2)

3 Synchronisation nebenläufiger Prozesse

Definition 16. :

- a) Ist das Ergebnis einer Ausführung nebenläufiger Prozesse von der zeitlichen Reihenfolge der Read- und Write-Operationen abhängig (**Race Condition**), so heißt sie **zeitkritische Ausführung**.
- b) Der Teil eines Programms, in dem lesend oder schreibend auf Variable, die mehreren Prozessen gemeinsam ist, zugegriffen wird, heißt **kritischer Bereich**.

Gesucht: Verfahren für **wechselseitigen Ausschluss** (**Mutual Exclusion**), d.h. Verfahren, die garantieren, dass sich zu jedem Zeitpunkt jeweils nur ein Prozess in einem kritischen Bereich befindet.

Anforderung: Kein Prozess soll unendlich lange warten, bis er einen kritischen Bereich betreten darf (**Starvation Free**).

3.1 Synchronisationshardware

Betrifft ein Prozess einen kritischen Bereich, darf er erst unterbrochen werden, wenn er diesen wieder verlassen hat.

Nachteile:

- nicht Starvation Free
- Verfahren wird nur für kurze Betriebssystemprozesse benutzt, nicht für Benutzerprozesse

3.2 Busy Waiting

Busy Waiting, oft auch **Spin Locking** genannt ist eine Softwarelösung zur Realisierung eines wechselseitigen Ausschlusses mit Hilfe einer Synchronisationsvariable

$$\text{flag} = \begin{cases} 0 & \text{kein Prozess im kritischen Bereich} \\ 1 & \text{kritischer Bereich belegt} \end{cases}$$

```
flag = 0          /* Initialwert */
...
while (flag != 0);      /* aktives Warten */
flag = 1;
critical_region();
flag = 0;
...
/* Kommt nicht in Frage!!! */
```

3.2.1 Lösungsverfahren für 2 Prozesse: Alternierender Zutritt

$$\text{turn} = \begin{cases} 0 & \text{Prozess A darf den kritischen Bereich betreten} \\ 1 & \text{Prozess B darf den kritischen Bereich betreten} \end{cases}$$

```
turn = 0;
```

Prozess A

```
...
while (turn != 0);
critical_region();
turn = 1;
...
```

Prozess B

```
...
while (turn != 1);
critical_region();
turn = 0;
...
```

alternierender Zugriff: A,B,A,B,A,B,...

Anwendungsfall 1 Producer, 1 Consumer mit einem Bufferplatz

Das Verfahren garantiert zwar wechselseitigen Ausschluss, jedoch ist nur striktes Alternieren möglich und durch das ständige Abfragen der `turn` Variable wird viel CPU-Zeit verschwendet.

Vorsicht: `while (turn != 0);`

Assembler:

→ Lade Wert von `turn`

Vergleich

Jump

Innerhalb der Schleife wird `turn` nicht verändert.

⇒ Mit Optimierung übersetzt wird der Wert von `turn` **nicht neu** geladen

⇒ `turn` muss als "volatile" deklariert werden

3.2.2 Peterson's Algorithmus (1981)

Für $i = 1, 2$

$$\begin{aligned} \text{interested_}_i &= \begin{cases} \text{true} & \text{Prozess } P_i \text{ will kritischen Bereich betreten} \\ \text{false} & \text{Prozess } P_i \text{ will kritischen nicht Bereich betreten} \end{cases} \\ \text{turn} &= \begin{cases} 1 & \text{Prozess } P_1 \text{ hat Vortritt} \\ 2 & \text{Prozess } P_2 \text{ hat Vortritt} \end{cases} \end{aligned}$$

Initialisierung:

```
turn = 1
interested_1 = false
interested_2 = false
```

Theorem 2. Der Peterson Algorithmus garantiert wechselseitigen Ausschluss ohne strenges Alternieren und ist fair.

Prozess P_1

```
interested_1 = true;
turn = 2; /* Z1 */
while (interested_2 && (turn = 2))
do skip;
critical_section();
interested_1 = false;
```

Prozess P_2

```
interested_2 = true;
turn = 1; /* Z2 */
while (interested_1 && (turn = 1))
do skip;
critical_section();
interested_2 = false;
```

Beweisskizze

1. Fall Eintrittswünsche P_1, P_2 sequentiell
2. Fall Eintrittswünsche P_1, P_1 sequentiell
3. Fall Eintrittswünsche P_1, P_2 nebenläufig

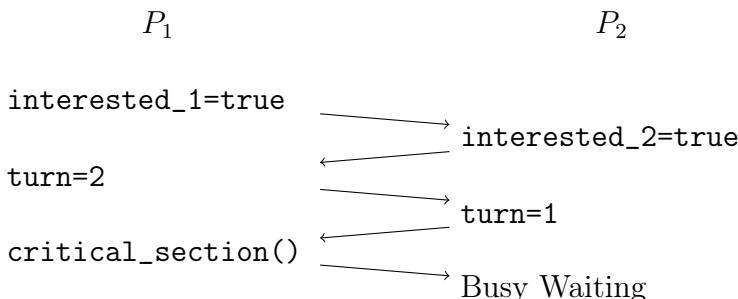
1. Fall $turn = 1$

```
interested_1 = interested_2 = false
interested_1 = true
turn = 2
P1 betritt den kritischen Bereich
interested_1 = false
P2 will kritischen Bereich betreten
... analog weiter
```

2. Fall nicht alternierend, keine Überprüfung nötig

3. Fall hier alle möglichen Verschachtelungen der Anweisungen in P_1 und P_2 prüfen

Vorüberlegung: Hier sein P_1 immer eine Anweisung vorraus:



Die Reihenfolge der `turn`-Anweisungen ($Z1, Z2$) entscheidet über die Reihenfolge der Zutritte. P_1, P_2 nebenläufige Zustandswünsche:

intrested_1	intrested_2	turn	P_1	P_2
true	true	1	Eintritt	Busy-Waiting
true	true	2	Busy-Waiting	Eintritt

3.3 Spin Locking mit Hardwareunterstützung

Wiederholung: Problem bei der ersten Softwarelösung für Spin Locking: (*siehe Busy Waiting*)

Viele (moderne) CPU-Architekturen unterstützen **atomare compare-and-exchange** (CMPXCHG) Instruktionen.

Die **atomare** Instruktion besteht aus:

```
cmpxchg (lock, _new, _old) {
    if (*lock == _old) {
        *lock := _new
        return TRUE
    } else {
        return FALSE
    }
}
```

Wozu kann man cmpxchg benutzen?

- cmpxchg ändert Werte im Speicher **atomar**
- cmpxchg ist ein nicht-priviligierte Kommando, das auch im user-mode ausgeführt werden kann.

⇒ Kann benutzt werden um **Locks** zu implementieren! (vorausgesetzt man weiß, wie man CPU-Instruktionen aus C/C++-Anwendungen heraus benutzen kann)

Beispiel:

```
cmpxchg(lock, 1, 0)

/* Falls lock==0 schreibe 1 and die Stelle von lock und liefere TRUE zurueck
 */
```

Synchronisation mittels Spin Locking mit CMPXCHG

Synchronisationsvariable:

$$\text{lockflag} = \begin{cases} 0 & \text{kein Prozess ist im kritischen Bereich} \\ 1 & \text{kritischer Bereich ist belegt} \end{cases}$$

```
lock(lockflag); /* aktives Warten */
critical_region();
unlock(lockflag);
...

typedef unsigned long lock_t;

void lock(lock_t * lock) {
    lock_t _old = 0;
    lock_t _new = 1;

    do {
        /* retry until lock is set to 0 (not used)
         * and changing to 1 succeeds
         */
    }
}
```

```

        } while ( ! cmpxchg ( lock , _new , _old ) );
}

void unlock ( lock_t * lock ) {
    *lock = 0;
}

```

3.4 Semaphor

Definition 17. Ein **Semaphor** ist eine Variable, auf die nur mit den nicht-unterbrechbaren (atomaren) Operationen DOWN und UP zugegriffen werden kann.

binäre Semaphore für wechselseitigen Ausschluss:

$$\text{Semaphor } S = \begin{cases} \leq 0 & \text{Prozess darf den kritischen Bereich nicht betreten} \\ = 1 & \text{Prozess darf den kritischen Bereich betreten} \end{cases}$$

Ablauf:

1. Prozess A will den kritischen Bereich betreten.
2. Prozess A fragt den Semaphorwert ab mit DOWN(S).
3. Falls $S = 0$ erfolgt eine synchrone Unterbrechung, d.h., A wartet im Zustand "blockiert", bis der kritische Bereich frei ist. Andernfalls betritt A den kritischen Bereich, d.h. A rechnet weiter.
4. Nach Verlassen des kritischen Bereichs wird der kritische Bereich mittels UP(S) wieder freigegeben.

Wechselseitiger Ausschluss:

```

...
DOWN(S);
critical_section();
UP(S);
...

DOWN(S) {
    S = S - 1;
    if (S < 0) {
        (der aufgerufene Prozess wird blockiert und in eine
         Warteschleife fuer S eingetragen);
    }
}

DOWN(S) {
    S = S + 1;
    if (Warteschlange von S nicht leer ist) {
        (erster Prozess in der Warteschlange wird geweckt);
    }
}

```

Vorteile:

- wechselseitiger Ausschluß,
 - UP, DOWN sind kurze Operationen, für die Nichtunterbrechbarkeit gerechtfertigt ist,
 - kein aktives Warten,
 - ermöglicht nicht nur wechselseitigen Ausschluß, sondern ist auch zur Synchronisation von Prozessabläufen geeignet (Prozesskoordination).

3.4.1 Producer-Consumer Problem

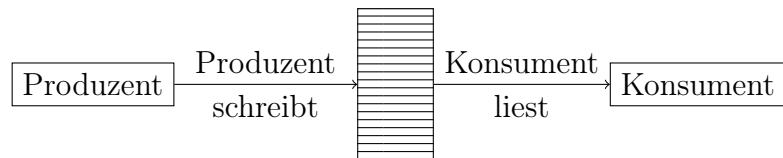


Abbildung 6: Aufbau Producer/Consumer mit n Pufferplätzen

1. Problem n Pufferplätze, die zyklisch belegt werden. Producer und Consumer dürfen sich nicht überholen!

2. Problem: $k \geq 1$ Producer, $l \geq 1$ Consumer

⇒ Komplexeres Problem als nur der Schutz eines kritischen Bereichs!

Lösung:

```

int n, next_w, next_r;
Semaphore mutex = 1;      // wechselseitiger Ausschluss
Semaphore free = n;       // Koordination: voller Buffer
Semaphore full = 0;       // Koordination: leerer Buffer

```

Consumer

```

while (True) {
    item = produce_item();

    DOWN(free);
    DOWN(mutex);

    write_item(item, next_w)
    next_w = next_w+1 modulo n;

    UP(mutex);
    UP(full));
}

```

D 1

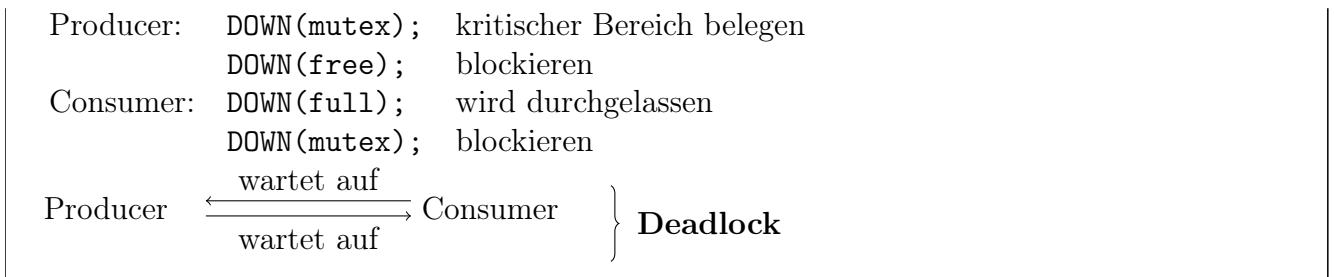
```
while(True) {
    DOWN(full);
    DOWN(mutex);

    item = remove_item(next_r);
    next_r = next_r+1 modulo n;

    UP(mutex);
    UP(free);

    consume(item);
}
```

Bemerkung Vertauschen der DOWN Operationen im Producer
Es sei $\text{free} = 0$, $\text{full} = n$



3.4.2 Readers-Writers Problem

Konkurrierender Zugriff auf Datei

Synchronisationsregeln

1. Beliebig viele Prozesse dürfen nebenläufig lesend auf die Datei zugreifen.
 2. Schreiben findet exklusiv statt.
 3. Falls ein Schreibwunsch eintrifft, wird kein neuer Lesezugriff zugelassen.

```
/* Gemeinsame Variablen */  
  
readcount = 0; /*Zaehler fuer Lesewuesche*/  
writecount = 0; /*Zahler fuer Schreibwuensche*/
```

Reader

```
...  
start_read();  
read(data);  
end_read();  
...
```

Writer

```
...  
start_write();  
write(data);  
end_write();  
...
```

Was ist zu überprüfen?

1. Gelten die Synchronisationsregeln?
 2. Verklemmungsfrei?

Beispiel

Lese und schreibprozesse treffen in folgender Reihenfolge ein:
 $R_1, R_2, R_3, W_1, R_4, R_5, W_2, R_6, \dots$

1. Möglichkeit: Nebenläufiges Lesen für hohen Durchsatz

- R_1, R_2, R_3 nebenläufiges Lesen
 - W_1 blockiert und wartet, Warteschleife [W_1]
 - R_4, R_5 nebenläufiges Lesen

- W_2 blockiert und wartet, Warteschleife [W_1, W_2]
- **Problem:** Wenn jetzt noch mehr Leseprozesse eintreten, dann würden die Schreibprozesse verhungern

2. Möglichkeit: Schreibpriorität, d.h. wenn ein Schreibwunsch vorliegt wird kein neuer Lesewunsch zugelassen

- R_1, R_2, R_3 nebenläufiges Lesen
- W_1 wartet bis Leseprozesse fertig
- W_1 schreibt
- R_4, R_5, W_2, R_6 nebenläufiges Lesen
- W_2 schreibt
- **Problem:** Leseprozesse können verhungern

Lösung:

```

readcount = 0;
writecount = 0;

Semaphor mutex_rc = 1; /* bin. Semaphor fuer Zugriff auf readcount */
Semaphor mutex_wc = 1; /* bin. Semaphor fuer Zugriff auf writecount */
Semaphor mutex_str = 1;

Semaphor sem_w = 1;      /* exklusiver Schreibzugriff */
Semaphor sem_r = 1; /* falls sem_r=1 dann Lesen erlaubt, falls sem_r=0 dann
                    Lesen nicht erlaubt (d. h. Schreibwunsch liegt vor) */

startread()
{
    DOWN(mutex_str);
    DOWN(sem_r);
    DOWN(mutex_rc);
    readcount++;
    if (readcount == 1) { DOWN(sem_w); }
    UP(mutex_rc);
    UP(sem_r); /* weitere reader zulassen */
    UP(mutex_str);
}

endread()
{
    DOWN(mutex_rc);
    readcount--;
    if (readcount == 0) { UP(sem_w); }
    UP(mutex_rc);
}

startwrite()
{
    DOWN(mutex_wc);
    writecount++;
    if (writecount == 1) { DOWN(sem_r); }
    UP(mutex_wc);
    DOWN(sem_w);
}

endwrite()
{
    UP(sem_w);
    DOWN(mutex_wc);
    writecount--;
    if (writecount == 0) { UP(sem_r); }
    UP(mutex_wc);
}

```

3.4.3 Nachteile von Semaphoren

- Ablaufsteuerung ist mit Semaphoren nicht intuitiv
- unübersichtliche Programmierung und dadurch Gefahr von fehlerhaftem Code mit Deadlock-Potential. Vor allem mit `goto` Anweisungen große Fehleranfälligkeit.
- Voraussetzung ist, dass Prozesse gemeinsamen Arbeitsspeicher für Semaphorrealisierung besitzen müssen (in verteilten Systemen nicht gegeben)

3.5 Monitor

Hoare 1974

hier: Version von Brinch Hansen, 1975

Definition 18. Ein Monitor ist ein Programmiersprachenkonstrukt, in dem alle Prozeduren und Datenstrukturen, die den Zutritt zum kritischen Bereich kontrollieren, gesammelt werden mit der Eigenschaft, daß nur ein Prozess zu einem Zeitpunkt in einem Monitor aktiv sein kann.

^ = Raum mit nur einem Schlüssel

"Bedingungsvariable" (condition) mit Operationen WAIT und SIGNAL zur **Koordination/Ablaufsteuerung**:

1. Kann eine Monitorfunktion in der Abarbeitung nicht fortfahren, ruft sie `WAIT(<condition name>)` auf, wodurch der aufrufende Prozess blockiert wird, d.h., der Prozess ist nicht mehr im Monitor aktiv.
Innerhalb eines Monitors können Prozesse auf unterschiedliche Ereignisse/Bedingungen warten.
2. Aufwecken von Prozessen:
`SIGNAL(<condition name>)`: Falls ein Prozess auf `<condition name>` wartet, wird dieser geweckt. Die Warteschlange wird gemäß FIFO verwaltet.
`SIGNAL`-Semantik: Es muß garantiert sein, daß der geweckte Prozess als nächster die Kontrolle über den Monitor erhält.
3. `WAIT` und `SIGNAL` dürfen nur innerhalb des Monitors benutzt werden.

Hoare: Der `SIGNAL` aufrufende Prozess wird blockiert und erst geweckt, wenn der fortgesetzte Prozess den Monitor verläßt.

Brinch Hansen: `SIGNAL` darf nur als letzte Anweisung in einer Monitor-Funktion stehen!!!

3.5.1 Monitorlösung für Producer-Consumer Problem

```
monitor producer_consumer {
    condition full, empty;
    integer count; /* Anzahl belegter Pufferplaetze */

    enter(item) {
```

```

        if (count == N) WAIT(full);
        write_item(item);
        count = count + 1;
        if (count == 1) SIGNAL(empty);

    remove(item) {
        if (count == 0) WAIT(empty);
        remove_item(item);
        count = count - 1;
        if (count == N-1) SIGNAL(full);
    }
}

```

3.5.2 Implementation von Monitoren

Spezieller Compiler, der Bibliotheksfunktionen einsetzt, falls

- Monitor betreten wird
- Monitor verlassen wird
- wait-Aufruf
- signal-Aufruf

Vorteil von Monitoren: Monitore erzwingen übersichtlichen Programmierstil, bei dem SSynchronisationscode "vom restlichen Programmtext getrennt wird.

3.5.3 Thread-Synchronisation in Java: Variante von Hoare's Monitorkonzept

3.6 Beispiel Pthread-Bibliothek

3.6.1 Pthread Mutex

Die Pthread-Bibliothek unterstützt **binäre** Semaphore, die sogenannten **Mutexe**:

Funktion	Beschreibung
pthread_mutex_init()	Erzeuge ein Mutex
pthread_mutex_lock()	Erlange die Sperre oder blockiere
pthread_mutex_trylock()	Erlange die Sperre oder erzeuge eine Fehlermeldung
pthread_mutex_unlock()	Gebe Sperre wieder frei
pthread_mutex_destroy()	Löschen/Aufräumen

Manual Pthread Mutex

PTHREAD_MUTEX(3)

Library Functions Manual

PTHREAD_MUTEX(3)

NAME

pthread_mutex_init, pthread_mutex_lock, pthread_mutex_trylock,
pthread_mutex_unlock, pthread_mutex_destroy - operations on mutexes

SYNOPSIS

```
#include <pthread.h>
```

```

pthread_mutex_t fastmutex = PTHREAD_MUTEX_INITIALIZER;

pthread_mutex_t recmutex = PTHREAD_RECURSIVE_MUTEX_INITIALIZER_NP;

pthread_mutex_t errchkmutex = PTHREAD_ERRORCHECK_MUTEX_INITIALIZER_NP;

int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *mutexattr);

int pthread_mutex_lock(pthread_mutex_t *mutex);

int pthread_mutex_trylock(pthread_mutex_t *mutex);

int pthread_mutex_unlock(pthread_mutex_t *mutex);

int pthread_mutex_destroy(pthread_mutex_t *mutex);

```

DESCRIPTION

A mutex is a MUTual EXclusion device, and is useful for protecting shared data structures from concurrent modifications, and implementing critical sections and monitors.

A mutex has two possible states: unlocked (not owned by any thread), and locked (owned by one thread). A mutex can never be owned by two different threads simultaneously. A thread attempting to lock a mutex that is already locked by another thread is suspended until the owning thread unlocks the mutex first.

`pthread_mutex_init` initializes the mutex object pointed to by `mutex` according to the mutex attributes specified in `mutexattr`. If `mutexattr` is `NULL`, default attributes are used instead.

The LinuxThreads implementation supports only one mutex attributes, the mutex kind, which is either “fast”, “recursive”, or “error checking”. The kind of a mutex determines whether it can be locked again by a thread that already owns it. The default kind is “fast”. See `pthread_mutexattr_init(3)` for more information on mutex attributes.

Variables of type `pthread_mutex_t` can also be initialized statically, using the constants `PTHREAD_MUTEX_INITIALIZER` (for fast mutexes), `PTHREAD_RECURSIVE_MUTEX_INITIALIZER_NP` (for recursive mutexes), and `PTHREAD_ERRORCHECK_MUTEX_INITIALIZER_NP` (for error checking mutexes).

`pthread_mutex_lock` locks the given mutex. If the mutex is currently unlocked, it becomes locked and owned by the calling thread, and `pthread_mutex_lock` returns immediately. If the mutex is already locked by another thread, `pthread_mutex_lock` suspends the calling thread until the mutex is unlocked.

If the mutex is already locked by the calling thread, the behavior of `pthread_mutex_lock` depends on the kind of the mutex. If the mutex is of the “fast” kind, the calling thread is suspended until the mutex is unlocked, thus effectively causing the calling thread to deadlock. If the mutex is of the “error checking” kind, `pthread_mutex_lock` returns immediately with the error code `EDEADLK`. If the mutex is of the “recursive” kind, `pthread_mutex_lock` succeeds and returns immediately, recording the number of times the calling thread has locked the

mutex. An equal number of `pthread_mutex_unlock` operations must be performed before the mutex returns to the unlocked state.

`pthread_mutex_trylock` behaves identically to `pthread_mutex_lock`, except that it does not block the calling thread if the mutex is already locked by another thread (or by the calling thread in the case of a “fast” mutex). Instead, `pthread_mutex_trylock` returns immediately with the error code `EBUSY`.

`pthread_mutex_unlock` unlocks the given mutex. The mutex is assumed to be locked and owned by the calling thread on entrance to `pthread_mutex_unlock`. If the mutex is of the “fast” kind, `pthread_mutex_unlock` always returns it to the unlocked state. If it is of the “recursive” kind, it decrements the locking count of the mutex (number of `pthread_mutex_lock` operations performed on it by the calling thread), and only when this count reaches zero is the mutex actually unlocked.

On “error checking” and “recursive” mutexes, `pthread_mutex_unlock` actually checks at run-time that the mutex is locked on entrance, and that it was locked by the same thread that is now calling `pthread_mutex_unlock`. If these conditions are not met, an error code is returned and the mutex remains unchanged. ‘‘Fast’’ mutexes perform no such checks, thus allowing a locked mutex to be unlocked by a thread other than its owner. This is non-portable behavior and must not be relied upon.

`pthread_mutex_destroy` destroys a mutex object, freeing the resources it might hold. The mutex must be unlocked on entrance. In the LinuxThreads implementation, no resources are associated with mutex objects, thus `pthread_mutex_destroy` actually does nothing except checking that the mutex is unlocked.

CANCELLATION

None of the mutex functions is a cancellation point, not even `pthread_mutex_lock`, in spite of the fact that it can suspend a thread for arbitrary durations. This way, the status of mutexes at cancellation points is predictable, allowing cancellation handlers to unlock precisely those mutexes that need to be unlocked before the thread stops executing. Consequently, threads using deferred cancellation should never hold a mutex for extended periods of time.

ASYNC-SIGNAL SAFETY

The mutex functions are not async-signal safe. What this means is that they should not be called from a signal handler. In particular, calling `pthread_mutex_lock` or `pthread_mutex_unlock` from a signal handler may deadlock the calling thread.

RETURN VALUE

`pthread_mutex_init` always returns 0. The other mutex functions return 0 on success and a non-zero error code on error.

ERRORS

The `pthread_mutex_lock` function returns the following error code on error:

EINVAL the mutex has not been properly initialized.

```

EDEADLK
    the mutex is already locked by the calling thread
    ("error checking" mutexes only).

```

The `pthread_mutex_trylock` function returns the following error codes on error:

```

EBUSY the mutex could not be acquired because it was currently
locked.

```

```
EINVAL the mutex has not been properly initialized.
```

The `pthread_mutex_unlock` function returns the following error code on error:

```
EINVAL the mutex has not been properly initialized.
```

```
EPERM the calling thread does not own the mutex ("error check-
ing" mutexes only).
```

The `pthread_mutex_destroy` function returns the following error code on error:

```
EBUSY the mutex is currently locked.
```

EXAMPLE

A shared global variable `x` can be protected by a mutex as follows:

```

int x;
pthread_mutex_t mut = PTHREAD_MUTEX_INITIALIZER;

```

All accesses and modifications to `x` should be bracketed by calls to `pthread_mutex_lock` and `pthread_mutex_unlock` as follows:

```

pthread_mutex_lock(&mut);
/* operate on x */
pthread_mutex_unlock(&mut);

```

LinuxThreads

PTHREAD_MUTEX(3)

3.6.2 Pthread Condition

Wie erreicht man damit Ablaufsteuerung, wie sie z.B. für das Producer-Consumer-Beispiel notwendig ist?

Die Pthread-Bibliothek bietet auch **Bedingungsvariable (Condition Variables)** an!

Funktion	Beschreibung
<code>pthread_cond_init()</code>	Erzeuge eine Bedingungsvariable
<code>pthread_cond_wait()</code>	Blockiere, um auf das Eintreffen der Bedingung zu warten
<code>pthread_cond_signal()</code>	Sende Signal an einen auf die Bedingung wartenden Thread
<code>pthread_cond_broadcast()</code>	Wecke alle Threads, die auf die Bedingung warten
<code>pthread_cond_destroy()</code>	Löschen/Aufräumen

Achtung: Wenn `pthread_cond_signal()` aufgerufen wird, aber kein Thread auf die Bedingung wartet, geht das Signal verloren.

Manual Pthread Conditions

- Wie beim Monitor-Konzept dürfen `pthread_cond_wait()` und `pthread_cond_signal()` nur innerhalb eines geschützten Raums, d.h. innerhalb eines geschützten kritischen Bereichs benutzt werden.
 - **Semantik von `pthread_cond_signal()`**
Warten mehrere Threads auf das Ereignis, wird derjenige Thread geweckt, der
 1. die höchste Priorität hat
 2. Falls alle Threads dieselbe Priorität haben, werden sie in FIFO-Reihenfolge geweckt
-

PTHREAD_COND(3)

Library Functions Manual

PTHREAD_COND(3)

NAME

`pthread_cond_init`, `pthread_cond_destroy`, `pthread_cond_signal`,
`pthread_cond_broadcast`, `pthread_cond_wait`, `pthread_cond_timedwait` –
operations on conditions

SYNOPSIS

```
#include <pthread.h>

pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

int pthread_cond_init(pthread_cond_t *cond, pthread_condattr_t
                      *cond_attr);

int pthread_cond_signal(pthread_cond_t *cond);

int pthread_cond_broadcast(pthread_cond_t *cond);

int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);

int pthread_cond_timedwait(pthread_cond_t *cond, pthread_mutex_t
                           *mutex, const struct timespec *abstime);

int pthread_cond_destroy(pthread_cond_t *cond);
```

DESCRIPTION

A condition (short for “condition variable”) is a synchronization device that allows threads to suspend execution and relinquish the processors until some predicate on shared data is satisfied. The basic operations on conditions are: signal the condition (when the predicate becomes true), and wait for the condition, suspending the thread execution until another thread signals the condition.

A condition variable must always be associated with a mutex, to avoid the race condition where a thread prepares to wait on a condition variable and another thread signals the condition just before the first thread actually waits on it.

`pthread_cond_init` initializes the condition variable `cond`, using the condition attributes specified in `cond_attr`, or default attributes if `cond_attr` is `NULL`. The LinuxThreads implementation supports no attributes for conditions, hence the `cond_attr` parameter is actually ignored.

Variables of type `pthread_cond_t` can also be initialized statically, using the constant `PTHREAD_COND_INITIALIZER`.

`pthread_cond_signal` restarts one of the threads that are waiting on the condition variable `cond`. If no threads are waiting on `cond`, nothing happens. If several threads are waiting on `cond`, exactly one is restarted, but it is not specified which.

`pthread_cond_broadcast` restarts all the threads that are waiting on the condition variable `cond`. Nothing happens if no threads are waiting on `cond`.

`pthread_cond_wait` atomically unlocks the mutex (as per `pthread_unlock_mutex`) and waits for the condition variable `cond` to be signaled. The thread execution is suspended and does not consume any CPU time until the condition variable is signaled. The mutex must be locked by the calling thread on entrance to `pthread_cond_wait`. Before returning to the calling thread, `pthread_cond_wait` re-acquires mutex (as per `pthread_lock_mutex`).

Unlocking the mutex and suspending on the condition variable is done atomically. Thus, if all threads always acquire the mutex before signaling the condition, this guarantees that the condition cannot be signaled (and thus ignored) between the time a thread locks the mutex and the time it waits on the condition variable.

`pthread_cond_timedwait` atomically unlocks mutex and waits on `cond`, as `pthread_cond_wait` does, but it also bounds the duration of the wait. If `cond` has not been signaled within the amount of time specified by `abstime`, the mutex mutex is re-acquired and `pthread_cond_timedwait` returns the error `ETIMEDOUT`. The `abstime` parameter specifies an absolute time, with the same origin as `time(2)` and `gettimeofday(2)`: an `abstime` of 0 corresponds to 00:00:00 GMT, January 1, 1970.

`pthread_cond_destroy` destroys a condition variable, freeing the resources it might hold. No threads must be waiting on the condition variable on entrance to `pthread_cond_destroy`. In the LinuxThreads implementation, no resources are associated with condition variables, thus `pthread_cond_destroy` actually does nothing except checking that the condition has no waiting threads.

CANCELLATION

`pthread_cond_wait` and `pthread_cond_timedwait` are cancellation points. If a thread is cancelled while suspended in one of these functions, the thread immediately resumes execution, then locks again the mutex argument to `pthread_cond_wait` and `pthread_cond_timedwait`, and finally executes the cancellation. Consequently, cleanup handlers are assured that mutex is locked when they are called.

ASYNC-SIGNAL SAFETY

The condition functions are not async-signal safe, and should not be called from a signal handler. In particular, calling `pthread_cond_sig-`

nal or pthread_cond_broadcast from a signal handler may deadlock the calling thread.

RETURN VALUE

All condition variable functions return 0 on success and a non-zero error code on error.

ERRORS

pthread_cond_init, pthread_cond_signal, pthread_cond_broadcast, and pthread_cond_wait never return an error code.

The pthread_cond_timedwait function returns the following error codes on error:

ETIMEDOUT

the condition variable was not signaled until the timeout specified by abstime

EINTR pthread_cond_timedwait was interrupted by a signal

The pthread_cond_destroy function returns the following error code on error:

EBUSY some threads are currently waiting on cond.

EXAMPLE

Consider two shared variables x and y, protected by the mutex mut, and a condition variable cond that is to be signaled whenever x becomes greater than y.

```
int x,y;
pthread_mutex_t mut = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
```

Waiting until x is greater than y is performed as follows:

```
pthread_mutex_lock(&mut);
while (x <= y) {
    pthread_cond_wait(&cond, &mut);
}
/* operate on x and y */
pthread_mutex_unlock(&mut);
```

Modifications on x and y that may cause x to become greater than y should signal the condition if needed:

```
pthread_mutex_lock(&mut);
/* modify x and y */
if (x > y) pthread_cond_broadcast(&cond);
pthread_mutex_unlock(&mut);
```

If it can be proved that at most one waiting thread needs to be waken up (for instance, if there are only two threads communicating through x and y), pthread_cond_signal can be used as a slightly more efficient alternative to pthread_cond_broadcast. In doubt, use pthread_cond_broadcast.

To wait for x to becomes greater than y with a timeout of 5 seconds,

do:

```
    struct timeval now;
    struct timespec timeout;
    int retcode;

    pthread_mutex_lock(&mut);
    gettimeofday(&now);
    timeout.tv_sec = now.tv_sec + 5;
    timeout.tv_nsec = now.tv_usec * 1000;
    retcode = 0;
    while (x <= y && retcode != ETIMEDOUT) {
        retcode = pthread_cond_timedwait(&cond, &mut, &timeout);
    }
    if (retcode == ETIMEDOUT) {
        /* timeout occurred */
    } else {
        /* operate on x and y */
    }
    pthread_mutex_unlock(&mut);
```

LinuxThreads

PTHREAD_COND(3)

Beispiel: Producer-Consumer-Beispiel ähnlich dem CubbyHole-Beispiel: Der Producer schreibt nacheinander die Werte 1 bis MAX in den gemeinsamen Buffer. Der Consumer leert den Buffer, indem er Null in den Buffer schreibt.

```
#include <stdio.h>
#include <pthread.h>
#define MAX 1000000000          /* Anzahl erzeugte Nummern */

pthread_mutex_t the_mutex;
pthread_cond_t condc, condp;      /* Bedingungsvariablen fuer Consumer und
                                 Producer */

int buffer = 0;                  /* Puffer zwischen Producer und Consumer */

void *producer(void *ptr){
    int i;

    for (i = 1; i <= MAX; i++) {
        pthread_mutex_lock(&the_mutex);           /* erlange exklusiven
                                                   Zugriff auf Puffer */
        while (buffer != 0) pthread_cond_wait(&condp, &the_mutex);
        buffer = i;                            /* lege Element in Puffer */
        pthread_cond_signal(&condc);           /* wecke Verbraucher ggf.
                                                   auf */
        pthread_mutex_unlock(&the_mutex);       /* gib Zugriff auf Puffer
                                                   frei */
    }
    pthread_exit(0);
}

void *consumer(void *ptr) {
    int i;
```

```

    for (i = 1; i <= MAX; i++) {
        pthread_mutex_lock(&the_mutex);           /* erlange exklusiven
                                                   Zugriff auf Puffer */
        while (buffer == 0) pthread_cond_wait(&condc, &the_mutex);
        buffer = 0;                            /* Puffer "leeren" */
        pthread_cond_signal(&condp);            /* wecke Erzeuger ggf. auf
                                                   */
        /*
        pthread_mutex_unlock(&the_mutex);       /* gib Zugriff auf Puffer
                                                   frei */
    }

    pthread_exit(0);
}

int main(int argc, char **argv) {
    pthread_t pro, con;
    pthread_mutex_init(&the_mutex, 0);
    pthread_cond_init(&condc, 0);
    pthread_cond_init(&condp, 0);
    pthread_create(&con, 0, consumer, 0);
    pthread_create(&pro, 0, producer, 0);
    pthread_join(pro, 0);
    pthread_join(con, 0);
    pthread_cond_destroy(&condc);
    pthread_cond_destroy(&condp);
    pthread_mutex_destroy(&the_mutex);
}

```

3.6.3 Pthread Read-Write-Locks

Funktion	Beschreibung
pthread_rwlock_init()	initialize a read-write-lock
pthread_rwlock_rdlock()	get a shared read lock
pthread_rwlock_wrlock()	get an exclusive write lock
pthread_rwlock_destroy()	destroy a read-write-lock
...	

Definition 19. Priority Inversion bezeichnet eine Scheduling-Anomalie: Ein Prozess niedriger Priorität wird gerechnet und ein Prozess höherer Priorität wartet auf ein Ereignis, daß nur der andere Prozess auslösen kann.

3.7 Nachrichtensystem

Nachricht (message)

Information, die von einem Prozess zu einem anderen Prozess übergeben wird.

Mailbox

Pufferplatz für Nachrichten, die abgeschickt, aber noch nicht empfangen wurden.

send (Mailbox-Adresse, Nachricht)

Kopiert Nachricht in die zur Zieladresse zugehörige Mailbox. Falls die Mailbox voll ist, wird blockiert gewartet, bis Platz in der Mailbox ist.

receive (Mailbox-Adresse, Nachricht)

Kopiert die Nachricht aus der Mailbox in den Adressraum des Empfängers, löscht die Nachricht in der Mailbox. Falls die Mailbox leer ist, wird blockiert gewartet, bis eine Nachricht eintrifft.

Semantik	blocking	non-blocking
send	Sender wartet, falls Mailbox voll ist (klassische Mailbox)	Sender kehrt mit Fehlermeldung zurück, falls Mailbox voll ist
receive	Falls Mailbox leer ist wird gewartet	Falls Mailbox leer ist, kehrt der Empfänger zurück

Mailbox Producer-Consumer

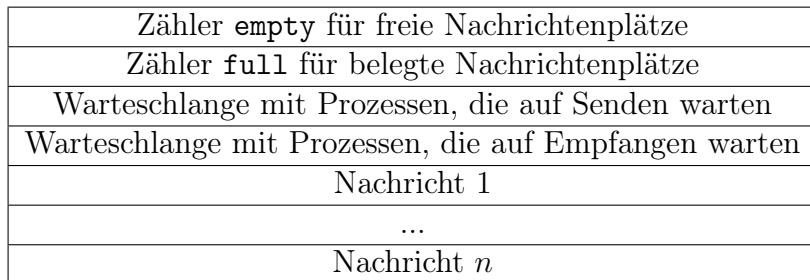
Producer

```
while (TRUE) {
    item=produce_item();
    send("ProducerConsumer", item);
}
```

Consumer

```
while (TRUE) {
    receive("ProducerConsumer", &item);
    consume_item(item);
}
```

Mailbox-Implementation



Mailbox-Adressierung ermöglicht:

- 1-1 Kommunikation
- n-1 Kommunikation
- 1-n Kommunikation bzw. Broadcast

Beispielimplementierungen:

1. UNIX lokal: `msgsnd`, `msgrcv`
2. blockierende UNIX-Sockets

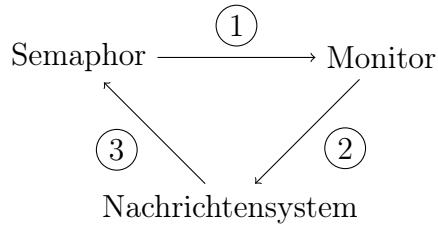
Vorteile:

- übersichtlicher als Semaphore,

- einfache "Realisierung als Systemaufrufe,
- getrennter Adressraum von Sender und Empfänger \Rightarrow weniger fehleranfällig,
- auch zur Kommunikation bzw. Synchronisation nicht-lokaler Prozesse geeignet

Theorem 3. Semaphore, Monitore und Nachrichtensysteme bieten die gleiche Mächtigkeit zur Prozesssynchroisation, d.h., alle Synchronisationsprobleme, die sich mit einem der 3 Modelle lösen lassen, sind auch mit den beiden anderen lösbar.

Beweis:

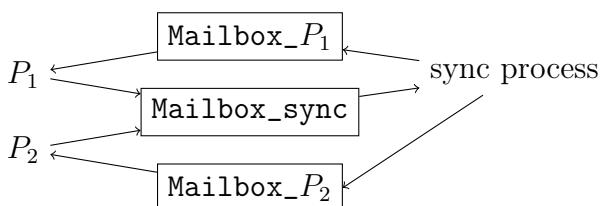


$\ddot{A} \rightarrow B": A$ lässt sich durch B realisieren

zu (1): *siehe Vorlesung*

zu (2): Sender \rightarrow Mailbox \rightarrow Empfänger
 Mailbox-Verwaltung $\stackrel{\wedge}{=}$ Producer-Consumer Problem (lässt dich mit Monitor lösen)

zu (3):
geg: send, receive
ges: Realisierung der Semaphorvariable UP, DOWN
Idee: Synchronisationsprozess



4 Speicherverwaltung

4.1 Swapping

Definition 20. Unter **Swapping** versteht man das Verlagern von Prozessen zwischen Hauptspeicher und Hintergrundspeicher.

Idee:

- **Auslagern (swap out)** blockierter Prozesse auf den Hintergrundspeicher.
- **Einlagern (swap in)** nach Übergang in den Zustand "bereit".

Probleme:

- Aufwand für Ein- und Auslagern,
- Belegungsstrategie beim Einlagern.

4.1.1 Belegungsstrategien

First Fit: Der Prozess wird in die erste hinreichend große Lücke eingelagert (Liste von vorn durchlaufen).

Next Fit: Analog zu First Fit von der aktuellen Zeigerposition aus.

Best Fit: Der Prozess wird in diejenige Lücke eingelagert, die den kleinsten Rest ergibt.

Worst Fit: Der Prozess wird in die größte Lücke eingelagert.

Ziel: Externe Fragmentierung (Externer Verschnitt), d.h. ungenutzten Hauptspeicherplatz, minimieren!

Buddy-Verfahren

Für Beispiel siehe Aufgabe 5.3: Speicherverwaltung
 n Listen für Lücken der Länge $1, 2, 4, 8, \dots, 2^{n-1}$

Belegen

1. Runde die Größe der Anforderung auf die nächste Zweierpotenz 2^k auf
2. Suche die erste nicht-leere 2^m -Liste mit $m \geq k$
3. Unterteile den ersten Block aus der 2^m -Liste!

Freigeben (Merge)

1. Falls sich der freigegebene Block auf seiner 2^k -Liste verschmelzen lässt, tue dies! Ggf. auf der 2^{k+1} -Liste wiederholen, usw.
- ⇒ reduziert externen Verschnitt.
- aber:** Es entsteht interner Verschnitt (interne Fragmentierung), da Prozeßgrößen i.a. keine 2er-Potenz sind.

4.1.2 Nachteil von Swapping

1. externe Fragmentierung

Abhilfe:

- Buddy-Verfahren \Rightarrow interne Fragmentierung
- Kompaktifizierung: Verlagern von Prozessen im Hauptspeicher, um den zusammenhängenden freien Adressraum zu vergrößern. \Rightarrow hoher Kopieraufwand!

2. Die Prozeßgröße wird durch die Hauptspeichergröße beschränkt!

4.1.3 Lokalitätsprinzip

Prozesse greifen während eines vorgegebenen Zeitintervalls nur auf eine Teilmenge ihres Adressraums zu.

\Rightarrow Diese Teilmenge ist durch eine höhere Zugriffswahrscheinlichkeit gekennzeichnet.

Fazit

Prozesse können laufen, ohne daß sich der komplette Adressraum im Hauptspeicher befindet \Rightarrow Gesamteinlagerung von Prozessen ist überflüssig!

4.2 Virtuelle Adressierung

Idee

Programme benutzen virtuelle Adressen!

Der virtuelle und der physikalische Adressraum werden in Blöcke gleicher Größe aufgeteilt. Üblich: Blöcke von 512 bytes – 8 KByte

Zur Laufzeit muß die virtuelle Adresse dann auf die physikalische Adresse im Hauptspeicher abgebildet werden.

Bezeichnungen

Ein Block im virtuellen Adressraum heißt **Seite (page)**, ein Block im physikalischen Adressraum heißt **Kachel (page frame)** bzw. Seitenrahmen.

Die **Seitentabelle** realisiert die Zuordnung: Seite \rightarrow Seitenrahmen

4.3 Paging-Verfahren

Definition 21. Ein **Seitenfehler** tritt auf, falls während der Programmausführung ein Zugriffswunsch auf eine Seite erfolgt, die sich nicht im Hauptspeicher befindet.

Demand-Paging := Seitenwechsel auf Anforderung

Gefahr: Seitenflattern: Das Betriebssystem beschäftigt sich nur noch mit Ein-/Auslagern von Seiten.

Definition 22. Ein Programmablauf wird durch eine Seitenzugriffsfolge, auch Referenzstring genannt, $W = s[1]s[2]\dots s[T]$ beschrieben, wobei $s[t] \in \mathbb{N}$ die Seitennummer der Seite, auf die zum Zeitpunkt t zugegriffen wird, bezeichnet.

Definition 23. $F(A, m, W)$ bezeichne die Anzahl Seitenfehler eines Paging-Algorithmus A zu gegebenem Referenzstring W und Hauptspeichergröße m.
Ein Paging-Algorithmus heißt gutartig, falls

$$\forall W \in \mathbb{N}^{\mathbb{N}} \quad \forall m \in \mathbb{N} : F(A, m + 1, W) \leq F(A, m, W)$$

Definition 24. $S(A, m, W, t)$ bezeichne die Menge der zum Zeitpunkt t gemäß dem Paging-Algorithmus A und dem Referenzstring W in den Hauptspeicher der Größe m eingelagerten Seiten.

Ein Algorithmus heißt Stack-Algorithmus, falls

$$\forall m \in \mathbb{N} \quad \forall t \in \mathbb{N} \quad \forall W \in \mathbb{N}^{\mathbb{N}} : S(A, m, W, t) \subseteq S(A, m + 1, W, t)$$

Verfahren

- $s[t]$ zum Zeitpunkt t nicht im Hauptspeicher \rightarrow einlagern!
- Hauptspeicher belegt \rightarrow Es muß eine Seite \bar{s} ausgewählt werden, die ausgelagert wird.
- Falls \bar{s} modifiziert ist $\rightarrow \bar{s}$ auf den Hintergrundspeicher zurückschreiben. Andernfalls kann \bar{s} überschrieben werden.

4.3.1 Ersetzungsstrategien

First-In-First-Out (FIFO)

älteste Seite wird ausgelagert. FIFO ist **nicht gutartig**. (**Belady's Anomalie**)

Least-Recently-Used (LRU)

Die am längsten nicht benutzte Seite wird ausgelagert. LRU ist **gutartig**.

Belady's optimaler Algorithmus

Die Seite mit dem größten Vorwärtsabstand wird ausgelagert.

Vorwärtsabstand der Seite x zum Zeitpunkt t eines Programms mit Laufzeit T :

$$d(x, t) := \begin{cases} k & \text{falls } x \notin \{s[t+1], \dots, s[t+k-1]\} \text{ und } s[t+k] = x \\ \infty & \text{falls } x \notin \{s[t+1], \dots, s[T]\} \end{cases}$$

Optimaler Algorithmus,
nur für die Beurteilung anderer Verfahren interessant, da $d(x, t)$ nicht a priori bekannt ist.

Working-Set-Modell (Denning 1970)

Definition 25. Gegeben sei der Referenzstring $s[1], s[2], \dots, s[t]$ eines Programms zum Zeitpunkt t . Dann heißt

$$W(t, T) := \{i \in \mathbb{N} \mid i \in \{s[t-T], s[t-T+1], \dots, s[t-1]\}\}$$

Working-Set (Arbeitsmenge) des Programms zum Zeitpunkt t .
 T := heißt Working-Set-Parameter.

Ein Working-Set-Algorithmus basiert auf folgender Eigenschaft:

Die Seiten des Working-Set werden im Hauptspeicher gehalten. Jede beliebige Seite, die nicht zum Working-Set $W(t, T)$ gehört, kann ausgelagert werden.

m	5	2	6	4	3	2	1	3	2	1
1	5	5	5	5	3	3	3	3	3	3
2		2	2	2	2	2	2	2	2	2
3			6	6	6	1	1	1	1	1
4				4	4	4	4	4	4	4
SF	x	x	x	x	x		x			

*6 SF
optimal!*

Abbildung 11: Beispiel: Working-Set

Statische/Dynamische Seitenersetzungsverfahren

Bei **statischen (lokalen) Verfahren** darf jeder Prozeß gleich viel Seitenrahmen bis zu einer vorgegebenen Maximalzahl im Hauptspeicher belegen.

Im Gegensatz dazu verändert sich bei **dynamischen (globalen) Verfahren** die Anzahl der Seitenrahmen, die ein Prozeß zur Laufzeit belegt. Es existiert keine maximale Obergrenze.

4.3.2 Lokalitätsprinzip für Demand-Paging

Prozesse greifen während eines vorgegebenen Zeitintervalls nur auf eine Teilmenge der Seiten ihres virtuellen Adressraums zu. → Diese Teilmenge ist durch eine höhere Zugriffswahrscheinlichkeit gekennzeichnet.

4.4 [BEISPIEL: UNIX!]

4.5 [BEISPIEL: WINDOWS NT!]

4.6 [BEISPIEL: LINUX!]

5 Dateisysteme

Das Dateisystem ist eine logische Abstraktion des genutzten physikalischen Speichers. Betriebs-systemsicht auf eine Datei: Folge von Bytes

Dateitypen

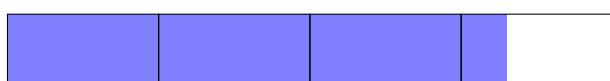
- Benutzerdateien: ASCII-Dateien, Binaries
- Systemdateien: Directories, Dateien zur Modellierung von I/O-Geräten wie z.B. Terminals und Platten

Magic number
Text Size
Data Size
BSS size
Symbol table size
Entry point
Flags
Text
Data
Relocation Bits
Symbol table

Abbildung 12: UNIX Executable Format a.out (aktuell: ELF-Format)

```
$ ll -s main.tex
4 -rw-r-r- 1 jakob users 2204 Feb 28 19:04 main.tex
```

Hinweis: -s zeigt die allokierte Größe der Datei in Blöcken an



Die Datei `main.tex` mit der Größe 2204 Byte belegt 4 Blöcke mit einer Größe von jeweils 512 Byte

Ansatz: Analog zum Hauptspeicher und dem virtuellen Adressraum eines Prozesses werden Dateien und Festplatte in Blöcke gleicher Größe eingeteilt.

Definition 26. Ein **Platten- bzw. Diskettenlaufwerk** (engl. **drive**) besteht aus einer oder mehreren Magnetplatten mit jeweils unter- oder oberseitigen **Lese/Schreibköpfen**. Die Daten werden auf Kreislinien auf der Oberfläche der Magnetplatten gespeichert, den sogenannten **Spuren** (engl. **tracks**). Die Spuren werden durchnummeriert. Die Spuren n aller Oberflächen heißen **Zylinder** n . Spuren werden in Abschnitte fester Datenlänge eingeteilt, den sogenannten **Sektoren**.

Beispiel:

ATA device, Model Number: SAMSUNG HD161HJ

Configuration:

Logical cylinders	max 16383
heads	16
sectors/track	63
Logical/Physical Sector size	512 bytes

seit ca. 2010 auch die Verwendung vom: Advanced Format mit 4096 Bytes großen Sektoren

5.1 Benutzerschnittstelle

Das Betriebssystem merkt sich zu jeder Datei Attribute wie z.B.

- Zugriffsrechte
- Eigentümer
- Erzeugungsdatum
- Zeitpunkt der letzten Änderung
- Dateigröße
- ...

Dateioperationen: `open()`, `read()`, `write()`, `close()`, `rm()`, Umbenennen, Attribute lesen und setzen.

OPEN(2)

Linux Programmer's Manual

OPEN(2)

NAME

`open`, `openat`, `creat` - open and possibly create a file

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *pathname, int flags);
...
```

DESCRIPTION

The `open()` system call opens the file specified by `pathname`. If the specified file does not exist, it may optionally (if `O_CREAT` is specified in `flags`) be created by `open()`.

The return value of `open()` is a file descriptor, a small, nonnegative integer that is used

in subsequent system calls (`read(2)`, `write(2)`, `lseek(2)`, `fcntl(2)`, etc.) to refer to the open file. The file descriptor returned by a successful call will be the lowest-numbered file descriptor not currently open for the process.

By default, the new file descriptor is set to remain open across an `execve(2)` (i.e., the `FD_CLOEXEC` file descriptor flag described in `fcntl(2)` is initially disabled); the `O_CLOEXEC` flag, described below, can be used to change this default. The file offset is set to the beginning of the file (see `lseek(2)`).

A call to `open()` creates a new open file description, an entry in the system-wide table of open files. The open file description records the file offset and the file status flags (see below). A file descriptor is a reference to an open file description; this reference is unaffected if pathname is subsequently removed or modified to refer to a different file. For further details on open file descriptions, see NOTES.

The argument flags must include one of the following access modes: `O_RDONLY`, `O_WRONLY`, or `O_RDWR`. These request opening the file read-only, write-only, or read/write, respectively.

In addition, zero or more file creation flags and file status flags can be bitwise-or'd in flags. The file creation flags are `O_CLOEXEC`, `O_CREAT`, `O_DIRECTORY`, `O_EXCL`, `O_NOCTTY`, `O_NOFOLLOW`, `O_TMPFILE`, and `O_TRUNC`. The file status flags are all of the remaining flags listed below. The distinction between these two groups of flags is that the file creation flags affect the semantics of the open operation itself, while the file status flags affect the semantics of subsequent I/O operations. The file status flags can be retrieved and (in some cases) modified; see `fcntl(2)` for details.

RETURN VALUE

`open()` returns the new file descriptor, or -1 if an error occurred (in which case, `errno` is set appropriately).

...

Linux

2017-09-15

OPEN(2)

Memory Mapped Files

Datei wird in den virtuellen Adressraum eines Prozesses eingebettet.

⇒ shared memory zwischen Prozessen

Beispiel: `mmap()`

MMAP(2)

Linux Programmer's Manual

MMAP(2)

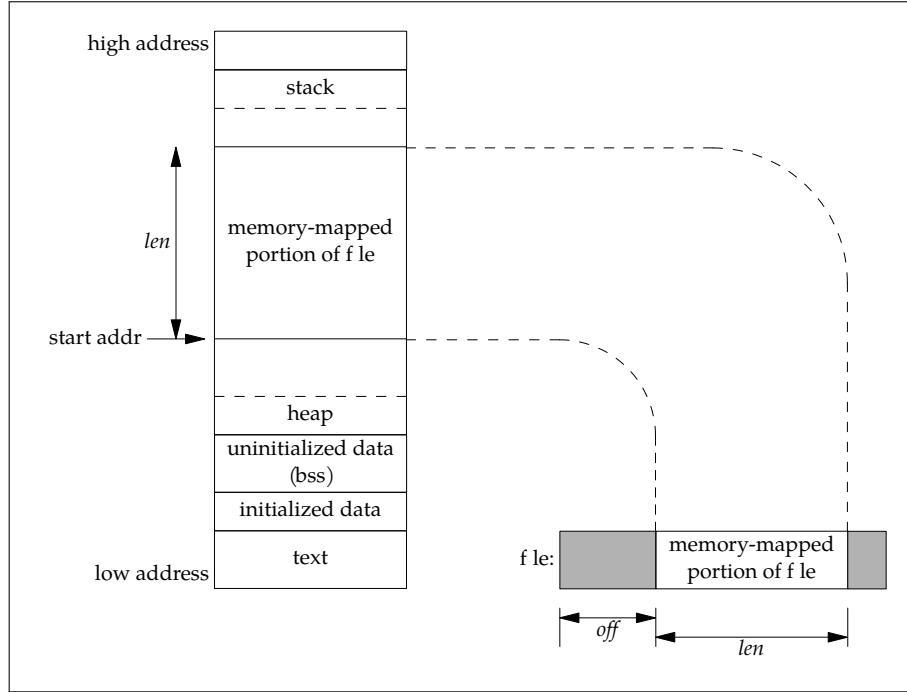
NAME

`mmap`, `munmap` - map or unmap files or devices into memory

SYNOPSIS

```
#include <sys/mman.h>

void *mmap(void *addr, size_t length, int prot, int flags,
```



Beispiel: memory-mapped file

```
int fd, off_t offset);
int munmap(void *addr, size_t length);
```

DESCRIPTION

`mmap()` creates a new mapping in the virtual address space of the calling process. The starting address for the new mapping is specified in `addr`. The `length` argument specifies the length of the mapping (which must be greater than 0).

If `addr` is `NULL`, then the kernel chooses the address at which to create the mapping; this is the most portable method of creating a new mapping. If `addr` is not `NULL`, then the kernel takes it as a hint about where to place the mapping; on Linux, the mapping will be created at a nearby page boundary. The address of the new mapping is returned as the result of the call.

The contents of a file mapping (as opposed to an anonymous mapping; see `MAP_ANONYMOUS` below), are initialized using `length` bytes starting at offset `offset` in the file (or other object) referred to by the file descriptor `fd`. `offset` must be a multiple of the page size as returned by `sysconf(_SC_PAGE_SIZE)`.

The `prot` argument describes the desired memory protection of the mapping (and must not conflict with the open mode of the file). It is either `PROT_NONE` or the bitwise OR of one or more of the following flags:

...

5.2 Implementation

Beispiel: 40 KB große Datei benötigt 10 Blöcke à 4KB

⇒ Wie soll die Datei auf dem Hintergrundspeicher abgespeichert werden?

1. Möglichkeit: Die Datei wird auf einem zusammenhängenden Plattenbereich abgespeichert.

2. Möglichkeit: Die Datei wird auf n Blöcke, die auf der Platte verteilt sein können, abgespeichert.

mode
owners (2)
timestamps (3)
size
12 direct blocks
single indirect
double indirect
triple indirect
block count
reference count
flags
generation number

UNIX i-node (index node)

Üblicherweise wird das Dateisystem mittels **Dateiverzeichnissen(Directories)** hierarchisch strukturiert. UNIX-Directories sind Dateien mit Einträgen von der Form:

i-node number		file name
---------------	--	-----------

Operationen auf Dateiverzeichnisse sind: `create`, `delete`, `opendir`, `closedir`, `readdir`, `rename`, `link`, `unlink`

5.3 Berkeley Fast File System

Ziel: Performance

Traditionelle Platteneinteilung im UNIX

Superblock enthält Dateisystem-Informationen wie z.B.

- Blockgröße
- Anzahl inodes
- Beginn der Free-Block-Liste

Verwaltung der freien Blöcke auf der Platte

1. Möglichkeit: Einfach verkettete Liste von Platten-Blöcken. Blöcke enthalten als Daten die Blocknummern von freien Blöcken.
2. **Möglichkeit:** Bitmap
Platte enthält n Blöcke \Rightarrow n Bit

$$\text{k-te Bit} = \begin{cases} 0 & \text{Block k ist belegt} \\ 1 & \text{Block k ist frei} \end{cases}$$

Optimierungen im Berkeley Fast File System

Hauptproblem: Lange Zugriffszeiten (engl. **Seek Time**) auf dem Speichermedium wie z.B. der Festplatte, bis der Lese-/Schreibkopf positioniert ist.

1. Änderung:

Seek Time verkürzen, indem inode und zugehörige Datei näher auf der Platte zusammengelegt werden \Rightarrow Platte wird in Cylinder Groups eingeteilt. Jede Cylinder Group hat eigenen Superblock, inodes, Bit Map für freie Blöcke in der **Cylinder Group** und Datenblöcke. Sofern möglich, werden Datenblöcke in demselben Zylinder abgespeichert wie der zugehörige inode.

2. Änderung

Erhöhen der Blockgröße

Forderung: Dateien 2^{32} Bytes sollen mit Double Indirect Blocks auskommen \Rightarrow Blockgröße $\geq 4\text{KB}$

5.4 Linux Dateisystem

Ext2 \approx Berkeleys Fast FS:

- Platte wird in "Block Groups" aufgeteilt. Jede Blockgruppe hat eigene Verwaltungsdaten.
- Bitmap für freie Blöcke
- Bitmap für freie i-nodes
- 12 direkte, 3 indirekte Blockadressen im i-node

Unterschiede: Blockgröße zwischen 1–4 KByte konfigurierbar, üblich sind 4 KByte keine Block/Fragment-Aufteilung

Ext3 ist eine Journaling-Filesystems:

Idee: Das Dateisystem führt Logbuch "Journal".

Schreiben eines Blockes:

1. Kopie des Blockes wird im Journal gespeichert
2. Block wird im Dateisystem geschrieben

3. Kopie im Journal löschen

Beispiel: Löschen einer Datei

1. Löschen der Datei aus ihrem Verzeichnis
2. I-Node aktualisieren ⇒ Reference-Count –
3. Falls Reference-Count == 0 ⇒ Freigeben des I-Nodes in den Pool der freien I-Nodes (zugehörige Bitmap aktualisieren)
4. Freigeben der zugehörigen Plattenblöcke (zugehörige Bitmap aktualisieren)

Im Journal kann notiert werden, bei welchem Arbeitsschritt das Dateisystem aktuell ist.
⇒ Schnellere Konsistenzchecks nach Systemabsturz.

Beispiel ext3 hat 3 Modi:

1. **Journal:** Alle Änderungen an Metadaten des Dateisystems und Dateien werden im Journal protokolliert. ⇒ sicherste, aber langsamste Modus
2. **Ordered:** Default-Einstellung, bei der nur die Änderungen an den Metadaten protokolliert werden. Zudem werden Datenblöcke vor den zugehörigen Metadatenänderungen auf Platte geschrieben.
3. **Writeback:** Nur die Änderungen an den Metadaten werden im Journal gemerkt. ⇒ schnellste Modus

Bemerkung: Journaling-Dateisysteme sind ReiserFS (erstes Journaling-Filesystems für Linux, aber nicht kompatibel zu ext2), ext3, xfs von SGI, JFS von IBM. Die übrigen Journaling-Dateisysteme arbeiten im **Writeback**-Modus.

Beispiel: ext4 (Default in Linux)

- Journaling-Dateisystem
- Unterstützung großer Dateisysteme und Dateien: Volumes bis 1 Exbibyte (EiB) und Dateien bis 16 Tebibytes (TiB) ⇒ von 32-Bit auf 48-Bit-Blockadressen umgestellt
- schneller Zugriff auf große Dateien: ext4 unterstützt sogenannte Extents (z.B. in JFS und XFS schon länger vorhanden), um Adressen von zusammenhängenden physikalischen Blocks effizienter abzuspeichern.
- Änderungen erfordern das Anpassen der i-node-Struktur

IEC 60027-2

International Electrotechnical Commission (IEC) spezifiziert in IEC 60027-2 2005, third Edition, Einheiten zur Basis 2 kibibytes, mebibytes, gibibytes, exbibytes, 2000

Faktor	Name	Symbol
2^{10}	kibi	Ki
2^{20}	mebi	Mi
2^{30}	gibi	Gi
2^{40}	tebi	Ti
2^{50}	pebi	Pi
2^{60}	exbi	Ei

5.5 Windows 2000: NTFS

- seit Windows NT
- Journaling-Filesystem
- organisiert Platte in "Volumes", Blockgröße kann pro Volume festgelegt werden: 512 Bytes–64 KByte, üblich sind 4 KByte
- statt i-nodes gibt es: Master File Table (MFT)
Jedes Volume besitzt eine MFT.
- Einträge/Records in der MFT à 1KB \gg | i-node |
- Anfangsadresse der MFT-Datei steht im Boot-Block.

Die ersten 16 MFT-Einträge sind Metadaten des Dateisystems:

Record	
0	: beschreibt MFT, z.B. Anfangsadresse
1	: Kopie von 0
2	: Logfile: Aktion (z.B. Create Directory, wird erst geloggt und dann ausgeführt)
3	: Volume-Beschreibung
4	: Definition der Datei-Attribute
5	: Root Directory
6	: Eintrag für die Datei, in der Bitmap für Blöcke gespeichert wird.
...	: ...
ab 16	: User File 1
...	: ...
...	: ...

Ein MFT-Record für eine Benutzerdatei oder ein Directory enthält alle beschreibenden Informationen (analog zum i-node):

- 13 mögliche Attribute
- Attribute werden als Tupel (type, length, value) gespeichert
- Standard-Attribute: Besitzer, Zeitstempel, ...
- Attribut: Dateiname (!)
- Attribut: Blocknummern der Blöcke, auf denen die Daten der Datei gespeichert sind
Abspeichern in Serien
- Dateien und Dateiverzeichnisse unter 1 KB können direkt auf dem MFT-Eintrag gespeichert werden!!!

Performanzvergleich ANNO 2017

Eckehart Synnatzschke: Benchmarking der Dateisysteme ext4, XFS, Btrfs und NTFS, Bachelorarbeit, Uni Potsdam, 2017.

- Messungen mit deaktiviertem Pagecache
- Benchmark: fio (Flexible I/O Tester) ist sowohl auf Linux als auch Windows lauffähig
- Blockgröße 4 KB
- Das Schreiben und Lesen erfolgt für die Dateigröße 1 GiB sowohl sequentiell als auch zufällig. Die Dateien der Größen 512 bis 1032 Byte werden nur sequentiell gelesen und geschrieben.
- Desktop-PC mit Debian GNU/Linux 8.8 Jessie, Windows 10Pro
- vollständige Versuchsbeschreibung siehe Bachelorarbeit

5.6 Sicherheit

Definition 27. Eine **Protection Domain** besteht aus einer Menge von Paaren (Objekt, Rechte), wobei Objekte z.B. Dateien, Drucker, ... sind und Rechte die Zugriffsrechte auf das Objekt beschreiben (lesend, schreibend, ausführbar) für die verschiedenen Domains.

Protection Matrix:

Domain	File1	File2	File3	File4	File5	File6	Printer1
1	r	r,w					
2		r	r,w,x	r,w			w
3						r,w,x	w
...							

Beispiel: In UNIX legen UID und GID die Protection Domains eines Prozesses fest.

1. Ansatz: Die Protection Matrix wird spaltenweise gespeichert:

- Zu jedem Objekt wird eine Liste, die sogenannte **Access Control List**, abgespeichert, die zu jeder Domain die Zugriffsrechte auf das Objekt enthält (nur nichtleere Einträge der Protection Matrix).
- Access Control Lists wurden in MULTICS implementiert.
- Unix benutzt ACLs mit drei Einträgen (Owner, Group, others).

Wiederholung: In UNIX legen UID und GID die Protection Domain eines Prozesses fest.

Beispiel: Darf ein Prozeß der Nutzerin `schnor` bzw. `pvogel` die folgende LaTeX-Datei öffnen?

```
$ ls -al folien.tex
-rw-r-r- 1 schnor users 10316 Dec 11 16:19 folien.tex
```

Unix Access Control Algorithm

1. Passt die effektive User-ID des Prozesses auf die Owner-ID? Falls ja, Owner-Rechte anwenden!
2. Passt die effektive Group-ID des Prozesses auf die Group-ID? Falls ja, Group-Rechte anwenden!
3. Ansonsten gelten die others-Rechte.

Ausnahme: `bettina@petzi 25 > 11 /usr/bin/passwd`
`-r-sr-xr-x 2 root wheel 26564 Nov 20 13:02 /usr/bin/passwd`

SUID-Bit: Führt ein Benutzer eine Programmdatei mit gesetztem Suid-Bit aus, so wird die **effektive uid (euid)** des ausführenden Prozesses durch die uid desjenigen Benutzers ersetzt, der Eigentümer der Programmdatei ist.

Bsp: `/bin/passwd, sendmail`

Merke: In UNIX legen euid und egid die Protection Domain eines Prozesses fest.

Erweiterte Access Control Lists unter Linux

- XFS unterstützt ACL, bei ext2, ext3 und ReiserFS optional,
- Samba ist ACL-fähig
- Jedes Objekt besitzt wie bisher unter UNIX Zugriffsrechte für den owner, group und others (**Minimal-ACL**).
- Der Minimal-ACL lassen sich weitere Einträge (**Access Control Entries (ACE)**) für **Named Users** und **Named Groups** hinzufügen. Es handelt sich dann um eine sogenannte **erweiterte ACL**.

Beispiel:

```
setfacl -m g:students:rw foo
```

gibt der Named Group **students** für die Datei **foo** Lese- und Schreibrechte.

- Eine erweiterte ACL benötigt eine **Rechtemaske**. Sie beschränkt die Zugriffsrechte für alle Gruppen (Owning Group und Named Groups) und Named Users, indem Maske und jeweilige Rechte bitweise mit der logischen Und-Operation verknüpft werden.

Beispiel:

```
setfacl -m m::r-x foo
```

entzieht allen Named Users und Named Groups das Schreibrecht.

Welcher ACE greift? - Matching ACE

1. Passt die effektive User-ID des Prozesses auf die Owner-ID?
Falls ja, Owner-ACE anwenden!
2. Passt die effektive User-ID des Prozesses auf die User-Id eines User-ACE?
Falls ja, User-ACE anwenden!
3. Gibt es Group-ACES, die auf den Benutzer/Prozeß passen?
Falls ja, gibt es darunter mindestens einen Group-ACE, der die angeforderten Rechte erlaubt?
 - Falls ja: Zugriff ist erlaubt!
 - Falls nein: Zugriff verboten.
4. Passen weder Nutzer- noch Gruppeneinträge, gelten die **others**-Rechte

Capabilities

2. Ansatz:

Die Protection Matrix wird zeilenweise gespeichert.

Domain	File1	File2	File3	File4	File5	File6	Printer1
1	r		r,w				
2		r		r,w,x	r, w		w
3						r,w,x	w

Zu jedem Prozeß wird eine Liste von Objekten, die sogenannte **Capability List**, mit Zugriffsrechten gespeichert.

Problem: Datei wird gelöscht ⇒ Capabilities für die Datei können auf mehreren Listen auftauchen.

Teil I

Grundlagen Rechnernetze

Teil II: Grundlagen Rechnernetze

- 1 Einführung Rechnernetze
 - 2 Das ISO-Referenzmodell OSI (Open System Interconnection)
 - 3 Die Sicherungsschicht
 - (a) Aufgaben der Sicherungsschicht
 - (b) Fehlererkennung und Fehlerkorrektur
 - (c) Flusssteuerung
 - (d) Beispiel: High-level Data Link Control (HDLC)
 - 4 Der IEEE-Standard 802 für lokale Netze
 - 5 Die Vermittlungsschicht, Beispiel Internetprotokoll (IP)
 - 6 Die Transportschicht, Beispiele UDP und TCP

6. Einführung Rechnernetze

1969 Arpanet

Geldgeber: (Defense) Advanced Research Projects Agency (DoD (Department of Defense), DoE (Department of Energy) und NASA)

Verbindung der Vielzahl verschiedener Netze \Rightarrow Internet Protocol (IP) (Mitte der 70er Jahre)

Literatur

- 1 Andrew S. Tanenbaum, David J. Wetherall: *Computernetzwerke*, Pearson Studium, 5. Auflage, 2012
 - 2 James F. Kurose, Keith W. Ross: *Computer Networking: A Top-Down-Ansatz*, Pearson, 7. Auflage, 2017.
 - 3 Douglas E. Comer: *TCP/IP - Studienausgabe: Konzepte, Protokolle, Architekturen*, Verlag mitp, 2011
 - 4 Kevin R. Fall und W. Richard Stevens: *TCP/IP Illustrated Volume 1: The Protocols*, Addison-Wesley, 2011.
 - 5 Gary R. Wright und W. Richard Stevens: *TCP/IP Illustrated Volume 2 - The Implementation*, Addison-Wesley, 1994

Prof. Bettina Schnor 373

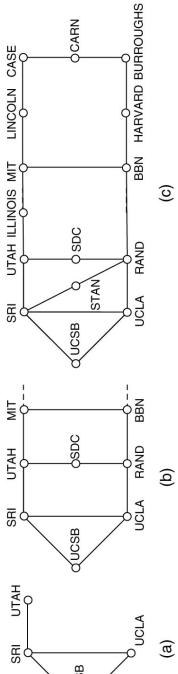
6. Einführung Rechnernetze

1969 Arpanet

Geldgeber: (Defense) Advanced Research Projects Agency (DoD (Department of Defense), DoE (Department of Energy) und NASA)

Verbindung der Vielzahl verschiedener Netze \Rightarrow Internet Protocol (IP) (Mitte der 70er Jahre)

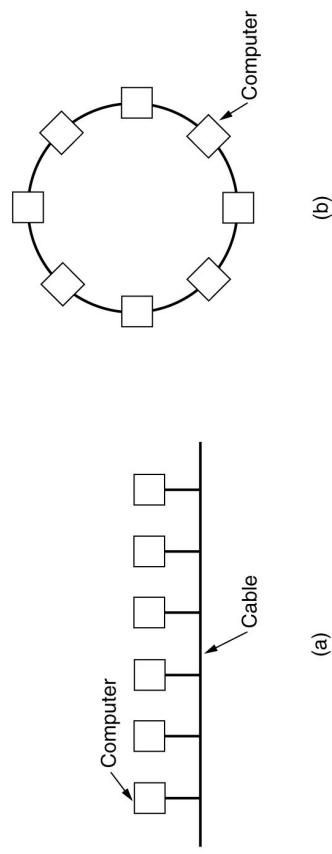
Prof. Bettina Schnorr 374



Arpanet: (a) Dezember 1969, (b) Juli 1970, (c) März, (d) April 1972, (e) September 1972
(Quelle: [2])

Local Area Networks (LAN)

- Durchmesser von wenigen Kilometern
- relativ hohe Übertragungsrate (10 MBit/s - 1 GBit/s)
- Topologien: Bus, Ring, Stern



Interprocessor distance	Processors located in same	Example
1 m	Square meter	
10 m	Room	
100 m	Building	Personal area network
1 km	Campus	Local area network
10 km	City	Metropolitan area network
100 km	Country	Wide area network
1000 km	Continent	
10,000 km	Planet	The Internet

Prof. Bettina Schnor

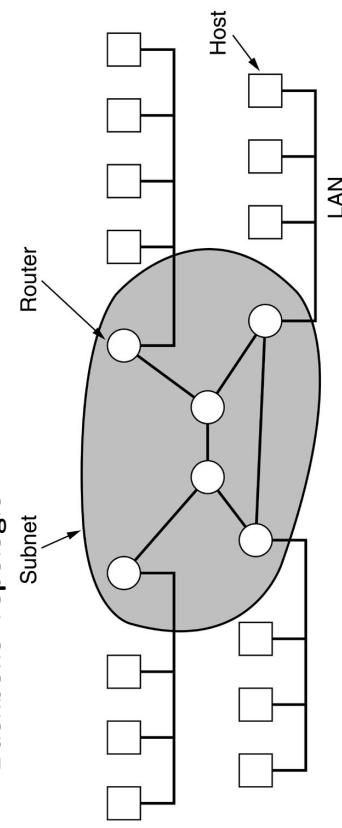
377

378

Metropolitan Area Networks (MAN)

- begrenzt auf Stadt/Region
- Vernetzung verschiedener LANs

Backbone-Topologie



Quelle: Tanenbaum

- Beispieldatechnologie: (ATM, FDDI) 1/10 Gigabit Ethernet

Prof. Bettina Schnor

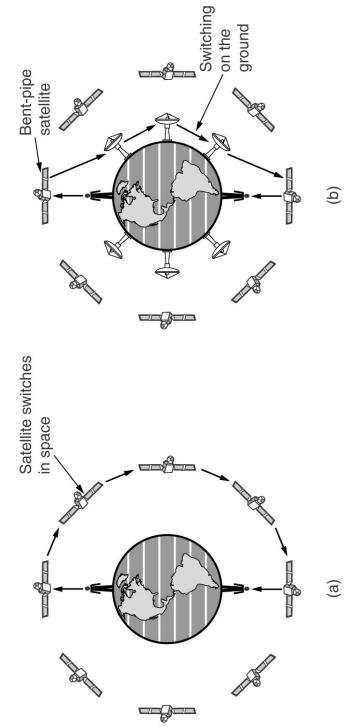
379

Prof. Bettina Schnor

378

Wide Area Networks (WAN)

- keine räumliche Begrenzung, ggf. Satellitenverbindungen



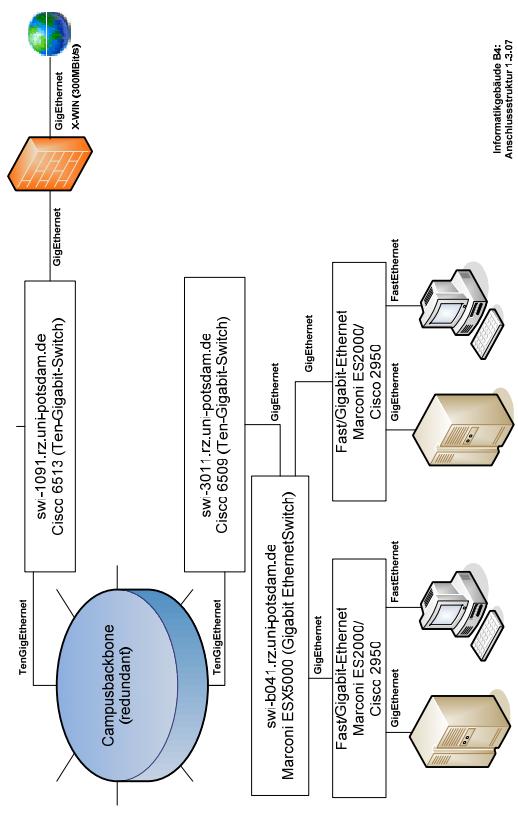
- Beispieldatechnologie: (ISDN, X.25) MPLS, Ethernet

- Beispiel: DFN

380

Prof. Bettina Schnor

Anschluß vom IFl an das Deutsche Forschungsnetz (DFN)



Prof. Bettina Schnorr

381

Prof. Bettina Schnor

३८३

Das Deutsche Forschungsnetz (DFN)

- 30.3.1984 Gründungsversammlung USA und UK sind ca. 4 Jahre voraus.
 - 9,6 Kilobit/s-Netz X.25 Wissenschaftsnetz, B-Win, G-Win, seit 2004 X-Win
 - 2000 war das DFN nach Transportvolumen und Zahl der Nutzer das größte Wissenschaftsnetz der Erde

Quelle: Editorial der DFN Mitteilungen, Ausgabe 86, Mai 2014, von Prof. em. Dr. Eike Jessen, Gründungsvorstand des DFN-Vereins

 - Anlässlich der Supercomputing Conference SC13 in Denver (17.-22.11.2013) nutzten Wissenschaftler erstmals einen **100Gigabit/s-Link** zwischen Deutschland und den USA (Austausch von Experimentdaten des Large Hadron Colliders in Genf).

Gründungsvorstand des DFN-Vereins

Gründungsvorstand des DFN-Vereins

תְּהִלָּה וְיִתְּנוּנָה תְּהִלָּה

„das“ DFN, versus „der“ DFN-Verein

Prof. Bettina Schnor

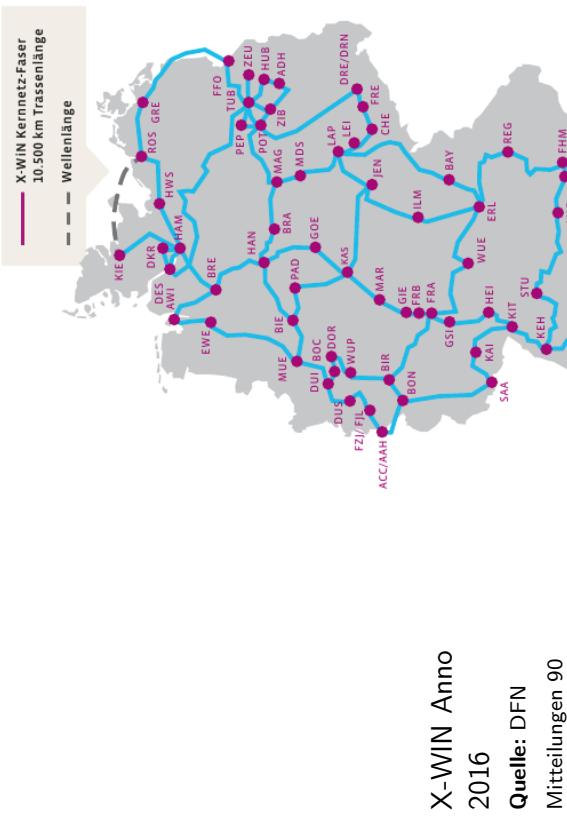
Prof. Bettina Schnor

383



- 1994 DFN ist Mit-Initiator der Gründung der Betreibergesellschaft DANTE für das europäische Wissenschaftsnetz GEANT (französisch ⇒ GEANT ist ein Backbone-Netzwerk, das rd. ca. 30 Forschungsnetze verbindet, darunter das GEANT ist auf 100Gigabit/s umgestellt.
- X-Win: Anschlusskapazität bis zu 100Gigabit/s 111 000 Kilometer Glasfaser

384



The map illustrates the X-WIN backbone network across Europe, featuring a dense network of nodes connected by a blue line representing 10,500 km of trunks. Nodes are labeled with three-letter abbreviations. A legend in the top left corner indicates that the solid blue line represents the X-WIN Kernnetz-Faser and the dashed blue line represents Wellenlänge.

X-WIN Kernnetz-Faser

10.500 Km Trassenlänge

Wellenlänge

Nodes shown include: KIE, DES, DKR, AWI, IWS, HAM, EWA, DRE, BIE, MUE, BOC, DOR, DUS, FZJ/FIL, ACC/AAH, BON, BIR, MAR, GIE, FRB, FRA, GS1, HEI, KAI, SAA, KIT, STU, KEH, BAY, ILM, WUE, ERL, REG, FHFM, CHE, FRE, DRE/DRN, LAP, LEI, IEN, HAN, BIA, PAD, GOE, GAS, ZIB, MAG, POT, TUB, PEI, ZEU, HUB, ADH, FFO.

X-WIN Anno 2016

Quelle: DFN

Mittelungen 90

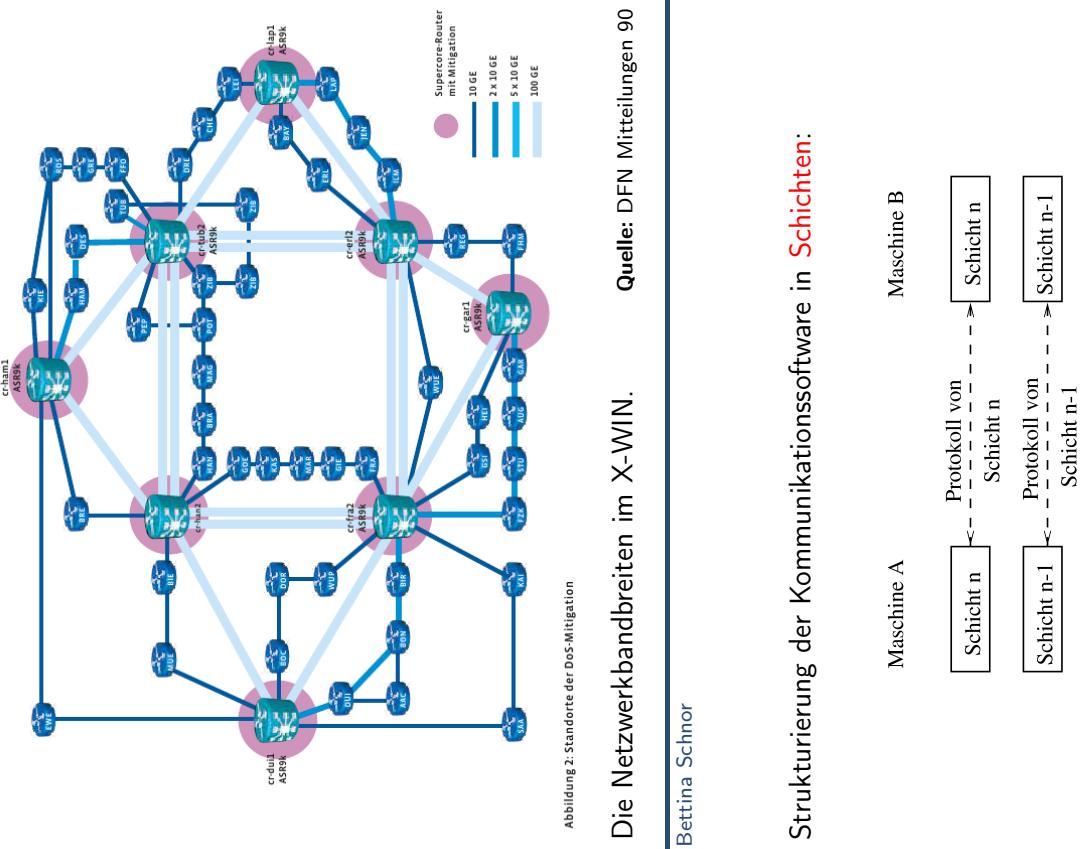
Übergang vom DEN in andere Netze:

- Géant: Frankfurt/Main (FRA), Hamburg HWS
 - DE-CLIX: Frankfurt / Main (FRA)
 - BCIX, ECIX

DE-CIX ist Deutschlands größter Internetknoten gemessen am Datendurchsatz.
Stand Juni 2017: weltweit größter Internetknoten gemessen am Datendurchsatz.

Die Netzwerkbandbreiten im X-WIN.
Quelle: DFN Mitteilungen 90

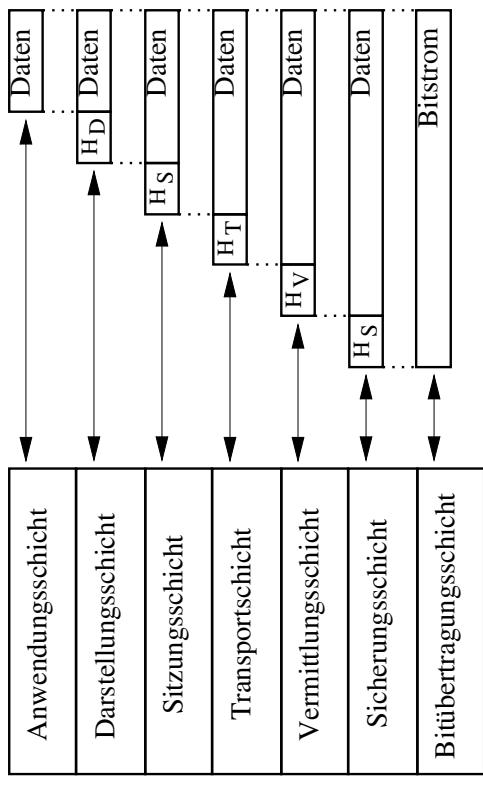
Strukturierung der Kommunikationssoftware in Schichten:



Ein Prozeß auf Schicht n auf Rechner A kommuniziert mit einem Partnerprozeß auf Schicht n auf Rechner B . Die Regeln, nach denen die Kommunikation abläuft, werden das Protokoll von Schicht n genannt.

Das ISO-Referenzmodell OSI

(International Standard Organization / Open Systems Interconnection)



Die Kommunikation der Partnerprozesse aus Schicht n wird durch den **Dienst** der Schicht $n - 1$ realisiert. Der Dienst beinhaltet eine Menge von Funktionen, die Prozessen aus Schicht n von der Schicht $n - 1$ angeboten werden.

Die Schichten und Protokolle bilden die **Netzarchitektur**.

1. Bitübertragungsschicht (Physical Layer)

Verschicken von Bitströmen
⇒ Normen für mechanische und elektrische Schnittstelle

2. Sicherungsschicht (Link Layer)

Die Dienstleistung der Bitübertragungsschicht ist die Übertragung einzelner Bits über ein (unzuverlässiges) Medium.
Die Sicherungsschicht ist für die **fehlerfreie** Übertragung von Bitketten zwischen benachbarten Rechnern verantwortlich.

Aufgaben:

- 1 Aufteilen der Daten in sogenannte **Datenübertragungsrahmen** (**Frame**) durch Einfügen von Rahmengrenzen.
- 2 Fehlererkennung und -korrektur
- 3 **Flusskontrolle**: Werden Nachrichten auf einem Link zwischen zwei Rechnern oder Vermittlungsknoten (Routern) verschickt, so kann es bei unterschiedlich leistungsfähigen Systemen zu einer Überlastung kommen und ggf. werden Pakete verworfen
⇒ Mittels **Flusskontrolle** wird die Senderate an die Verarbeitungsrate angepasst.

1. Datenübertragungsrahmen

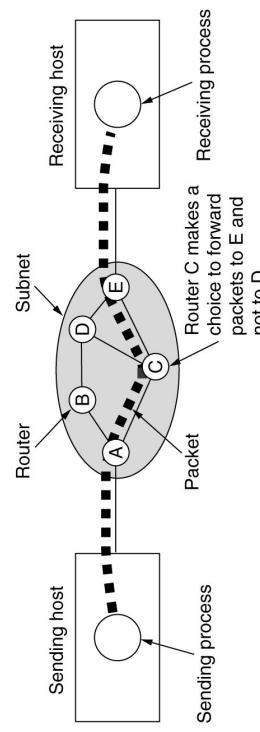
Beispielprotokoll: HDLC (High level Data Link Control, ISO)

3. Vermittlungsschicht (Network Layer)

Bisher: Wir können nur Nachrichten zwischen direkt verbundenen Rechnern schicken.

Die Vermittlungsschicht ist verantwortlich für den Transport von Nachrichten von einem Quellrechner zu einem Zielrechner über beliebig, untereinander verbundene Datenleitungen und Teilnetze.

Die Nachrichten müssen an den Zwischenstationen **vermittelt** werden.



(Quelle: Tanenbaum)

Prof. Bettina Schnor

393

Prof. Bettina Schnor
394

Es lassen sich zwei Dienstarten unterscheiden:

- **Virtual Circuit Service (verbindungsorientierter Dienst):** Es wird zwischen Quell- und Zielrechner eine „perfekte“ **logische (virtuelle)** Verbindung aufgebaut und am Ende der Kommunikation wieder abgebaut.

Beispiel: X.25 PLP (Packet Layer Protocol),
X.25 PLP ist das Schicht 3 Protokoll der CCITT-Empfehlung
X.25 für öffentliche Netze

Who's who?

Comité Consultatif International de Télégraphique et Téléphonique (Behörde der Vereinten Nationen)

X.25 ist eine Protokollfamilie für WANs über das Telefonsystem (Datex-P der deutschen Telekom): Anbindung von Alarm- und Notrufleitstellen sowie von EC-Kartenautomaten

Aufgaben:

- Wegwahl (**Routing**)
- Anpassung bei Übergang an ein anderes Netzwerk (Adressierung, Paketgröße, ...)
- Flußkontrolle
- Abrechnung der Dienstleistung (**Accounting**)

Prof. Bettina Schnor
394

- **Datagram Service (verbündungloser Dienst):** Die einzelnen Datenpakete einer Nachricht werden unabhängig voneinander vom Quell- zum Zielrechner übertragen. Jedes Datenpaket enthält Quell- und Zieladresse.
- **Beispiel:** Internetprotokoll (IP)

Prof. Bettina Schnor

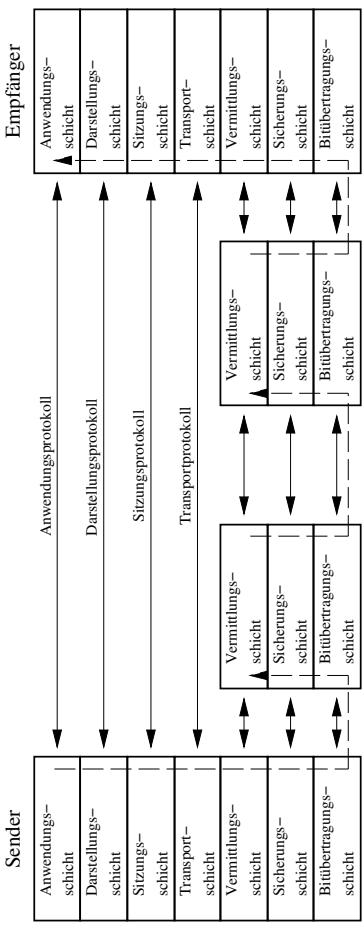
395

Prof. Bettina Schnor
396

Kommunikationsfluß

4. Transportschicht (Transport Layer)

Das Transportprotokoll ist ein **Ende-zu-Ende-Protokoll** zwischen Quell- und Zielrechner.



Prof. Bettina Schnor
397

5. Sitzungsschicht (Session Layer)

Bereitstellung von Diensten, die von Anwendungen häufig benötigt werden. \Rightarrow Unterstützung von sogenannten **Sitzungen**

Weitverbreitete Meinung in den 90ern: „Die Sitzungsschicht ist Erfindung der ISO!“

Beispiele für OSI-Dienste:

- **Checkpoints** für zuverlässige Datenübertragung,
 - Beispiel: openFT von Fujitsu (Einsatzgebiet: Behörden- und Bankensektor),
 - **Token-Management** für Videokonferenzen
 - **Client-Server-Kommunikation: Remote Procedure Call**
 - RPC-Protokolle für **synchrone** Kommunikation zwischen Client und Server

Prof. Bettina Schnor
398

6. Darstellungsschicht (Presentation Layer)

Wesentliche Aufgabe der Transportschicht ist die Erhöhung der Dienstqualität des Vermittlungsdienstes:

- ggf. Paketverlust überwachen und korrigieren,
- ggf. Auf- und Abbau von Transportverbindungen
- Flußkontrolle zwischen Quell- und Zielrechner
- ggf. Aufteilen der Nachricht in kleinere Dateneinheiten für Schicht 3
- Multiplexing oder Splitting auf Verbindungen der Schicht 3

Beispiele: TCP (Transmission Control Protocol),
UDP (User Datagram Protocol)

Prof. Bettina Schnor
398

```
htonl, htons, htonl, htons -- convert values between host  
and network byte order
```

LIBRARY

Standard C Library (libc, -lc)

These routines convert 16 and 32 bit quantities between network byte order and host byte order.
(Network byte order is big endian, or most significant byte first.)

On machines which have a byte order which is the same as the network order, routines are defined as null macros.

These routines are most often used in conjunction with Internet addresses and ports as returned by `gethostbyname(3)` and `getservent(3)`.

7. Anwendungsschicht (Application Layer)

In der Anwendungsschicht laufen die Benutzerprogramme (Anwendungen).

Beispiele: Electronic Mail, Dateitransfer (ftp, http get/post), Online-Banking, eCommerce, ...

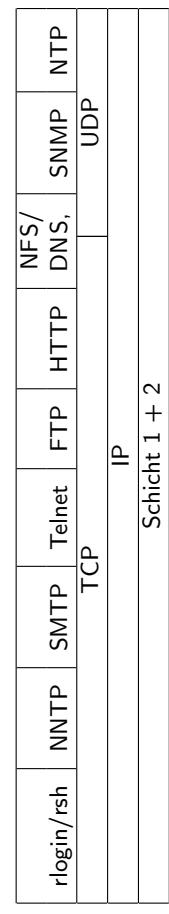
Otto Spaniol (1996): „Die neue ICE-Zeit!“

Prof. Bettina Schnor

401

402

Internet Protocol Stack



NNTP (Network News Transfer Protocol)
SMTP (Simple Mail Transfer Protocol)
DNS (Domain Name Service)
SNMP (Simple Network Management Protocol)
NTP (Network Time Protocol)
NFS (Network File System)

Key to Internet Success: Layers

Applications

...built on...
Reliable (or unreliable) transport

...built on...
Best-effort global packet delivery

...built on...
Best-effort local packet delivery
...built on...
Physical transfer of bits

Prof. Bettina Schnor

Prof. Bettina Schnor

Quelle: Prof. Scott Shenker, UC Berkeley

403

404

sendmail - Anekdoten

Eric Allman ist Autor von sendmail, syslog und steht offen zu seiner Homosexualität:

Es ist eine Art perverse Befriedigung zu wissen, dass es im Grunde unmöglich ist, Hassmails gegen Schwule durch das Internet zu schicken, ohne dass diese von einem schwulen Programm berührt wurden.



Quelle: Wikipedia, Bild Ilya Schurov

Internet Engineering Task Force (IETF)

<http://www.ietf.org/>
Die IETF verfaßt die Internet Standards, die sogenannten Requests for Comments (RFCs).

Es ist eine Art perverse Befriedigung zu wissen, dass es im Grunde unmöglich ist, Hassmails gegen Schwule durch das Internet zu schicken, ohne dass diese von einem schwulen Programm berührt wurden.

Gutes Intro zur Arbeitsweise der IETF in:
Christian Friebe: *IPv6 im Wandel - Analyse der IPv6-Standardisierung unter Sicherheitsaspekten*, Masterarbeit (Lehramt), Universität Potsdam, 2014.

Prof. Bettina Schnor

405

406

Beispiel: RFC 2119

Network Working Group
Request for Comments: 2119
BCP: 14
Category: Best Current Practice⁷

S. Bradner
Harvard University
March 1997

Abstract

In many standards track documents several words are used to signify the requirements in the specification. These words are often capitalized. This document defines these words as they should be interpreted in IETF documents. Authors who follow these guidelines should incorporate this phrase near the beginning of their document:

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Key words for use in RFCs to Indicate Requirement Levels

Status of this Memo

This document specifies an Internet Best Current Practices for the Internet Community, and requests discussion and suggestions for improvements. Distribution of this memo is unlimited.

⁷Weitere Kategorien: Informational, Standards

Note that the force of these words is modified by the requirement level of the document in which they are used.

1. **MUST** This word, or the terms “REQUIRED” or “SHALL”, mean that the definition is an absolute requirement of the specification.
2. **MUST NOT** This phrase, or the phrase “SHALL NOT”, mean that the definition is an absolute prohibition of the specification.
3. **SHOULD** This word, or the adjective “RECOMMENDED”, mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
4. **SHOULD NOT** This phrase, or the phrase “NOT RECOMMENDED” mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

5. **MAY** This word, or the adjective “OPTIONAL”, mean that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein, an implementation which does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)
6. **Guidance in the use of these Imperatives**
Imperatives of the type defined in this memo must be used with care and sparingly. In particular, they MUST only be used where it is actually **required for interoperability** or to limit behavior which has potential for causing harm (e.g., limiting retransmissions) For example, they must not be used to try to impose a particular method on implementors where the method is not required for interoperability.

3 Weisheiten zur Protokoll- und Netzwerkstandarisierung

7. Security Considerations

These terms are frequently used to specify behavior with security implications. The effects on security of not implementing a MUST or SHOULD, or doing something the specification says MUST NOT or SHOULD NOT be done may be very subtle. Document authors should take the time to elaborate the security implications of not following recommendations or requirements as most implementors will not have had the benefit of the experience and discussion that produced the specification.

„If you know what you are doing, 3 layers are enough. If you don't, even 17 won't help.“

(Mike Padlipsky, *The Elements of Network Style*, Prentice-Hall, 1985)

„The nice thing about standards is that there are so many to choose from.“

(Andrew Tanenbaum, 1988)

„Network standards are like sausages: If you intend to use them, you really don't want to see how they are made.“

(Gerald Cole, University of California, Los Angeles (UCLA), in Seifert: *Gigabit Ethernet*)

Der Erlrouter

Wer routet so spät durch Nacht und Wind?
Es ist der Router, er routet geschwind!
Bald routet er hier, bald routet er dort
Jedoch die Pakete, sie kommen nicht fort.

Finster der Tunnel, die Bandbreite knapp,
wie schön war die Backplane im eigenen Hub.
Am Ende des Tunnels: Das Päckchen ist weg,
vernichtet vom Cyclic Redundancy Check.

Quelle: <http://www.iks-jena.de/mitarb/lutz/usernet/Rainer.html>

Quelle: <http://www.iks-jena.de/mitarb/lutz/usernet/Rainer.html>

501

9. Vermittlungsschicht (Network Layer)

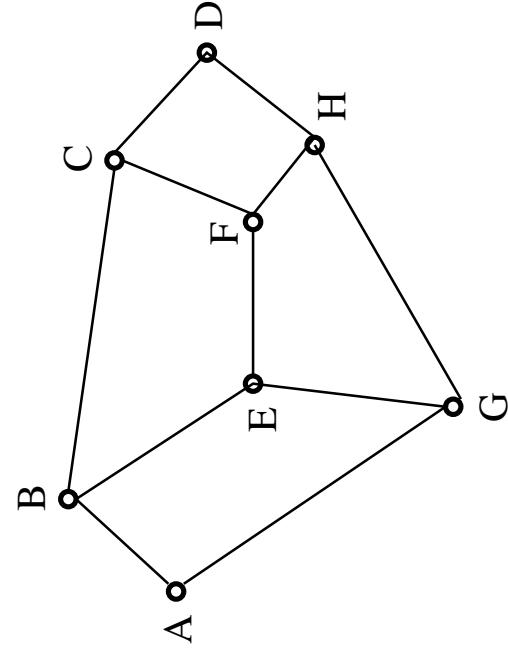
9.1 Wegwahlverfahren (Routing) und Addressierung

501

Prof. Bettina Schnor

501

Prof. Bettina Schnor



Bisher: Wir können nur Nachrichten zwischen direkt verbundenen Rechnern schicken: Entweder Punkt-zu-Punkt verbunden oder Broadcastmedium im LAN.

Die Vermittlungsschicht ist verantwortlich für den Transport von Nachrichten von einem Quellrechner zu einem Zielrechner über beliebig, untereinander verbundene Datenleitungen und Teilnetze. Die Nachrichten müssen an den Zwischenstationen **vermittelt** werden.

9.1 Wegwahl (Routing) und Addressierung

9.2 Fragmentierung

9.3 Beispiel: Internet Protocol (IP) und ICMP

Ziele eines Wegwahlverfahrens:

- 1 Minimalinformation eines Knotens: Adresse der Nachbarknoten
- Fluten: Das Paket wird auf allen Leitungen mit Ausnahme der Empfangsleitung gesendet.

 - + einfaches Verfahren
 - + Paket findet den Weg zum Empfänger

- 1 Übertragungszeit minimieren, d.h. den „schnellsten“ Weg vom Sender zum Empfänger bestimmen,

- Fluten: Das Paket wird auf allen Leitungen mit Ausnahme der Empfangsleitung gesendet.

 - + einfaches Verfahren
 - + Paket findet den Weg zum Empfänger

 - + Der schnellste Weg hängt ab von:
 - Anzahl Vermittlungsrechner auf dem Weg (Anzahl **Hops**), da in jedem Rechner Verarbeitungszeit anfällt.
 - Leitungslänge und Bandbreite,
 - Auslastung der Vermittlungsrechner.

- 2 Gesamtdurchsatz maximieren,
- 3 Übertragungskosten minimieren.

Klassifikation von Wegwahlverfahren

- 1 **statische Wegwahlverfahren:** Wege werden im voraus berechnet und in sogenannten **Wegwahltabellen** gespeichert.

Mögliche Parameter:

- Netztopologie
- Leitungslänge und Bandbreite
- erwartete Auslastung der Leitungen

Ein **Link-State-Routing-Algorithmus** setzt die komplette Kenntnis des Netzwerk-Graphen voraus.
⇒ Jeder Router kann die kürzesten Wege berechnen und damit seine Wegwahltabelle bestimmen.

Bestimmung des kürzesten Weges \Rightarrow Dijkstra-Algorithmus

- 2 **adaptive Wegwahlverfahren:** Wegwahl passt sich der aktuellen Netzsituation an:
 - Störungen (Leitungs-/Vermittlungsrechnerausfall)
 - aktuelle Lastsituation

Da volle Kenntnis des Netzwerk-Graphen vorausgesetzt wird, ist dieses Routingverfahren meist nur innerhalb des Netzes eines Netzwerkbetreibers anwendbar.

Dijkstra-Algorithmus zur Bestimmung des kürzesten Weges (Wiederholung)

Initialisierung:

Die Netztopologie entspricht einem zusammenhängenden Graphen $G = (E, K)$, wobei die Eckenmenge E die Menge der Vermittlungsrechner und die Kantenmenge K die Leitungen darstellt.

Der Graph ist bei Simplexbetrieb gerichtet, bei Duplexbetrieb ungerichtet. Im folgenden wird stets Duplexbetrieb betrachtet.

Aufgabe: Gesucht wird der kürzeste Weg von Vermittlungsrechner v_0 zu allen übrigen Knoten des Netzes.

Die Länge der Kanten wird mittels einer **Gewichtsfunktion** f gemessen:

$$f : E \times E \rightarrow R^+$$

Die **Distanzfunktion** gibt die Länge eines Weges v_0, v_1, \dots, v_k an:

$$d(v_0, v_k) = f(v_0, v_1) + f(v_1, v_2) + \dots + f(v_{k-1}, v_k)$$

Prof. Bettina Schnor

508

$S := \{v_0\}$ Menge derjenigen Knoten, zu denen ein kürzester Weg bereits gefunden wurde

Bestimme für alle $v \in E$ die Distanzfunktion

$$d(v) := \begin{cases} 0 & \text{falls } v = v_0 \\ f(v_0, v) & \text{falls die Kante } (v_0, v) \text{ existiert} \\ \infty & \text{sonst} \end{cases}$$

Prof. Bettina Schnor

509

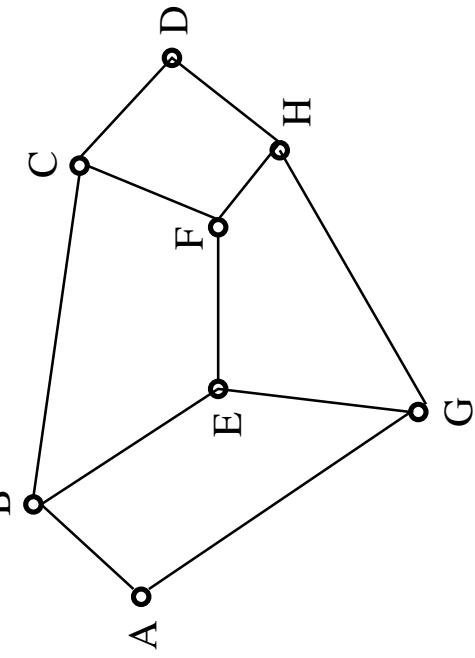
Bemerkung:

1. Der Algorithmus findet kürzeste Wege zu allen Knoten des Netzes.
2. Komplexität des Algorithmus: $O(n^2)$
3. In der Praxis wird als **Gewichtsfunktion** für die Kanten: $f = 1$ gewählt, d.h. es werden in der Distanzfunktion die Anzahl Hops gezählt.
4. $v_0, v_1, v_2, \dots, v_k, w$ sei ein kürzester Weg von v_0 nach w .
Dann ist v_1, v_2, \dots, v_k, w ein kürzester Weg von v_1 nach w .

Folgerung:

Statt sich den kompletten Weg zum Zielknoten zu merken, genügt es sich einen Nachbarknoten zu merken, der auf einem kürzesten Weg zum Zielknoten liegt.
 \Rightarrow kurze Wegwahltabellen

Beispiel: Wegwahltabelle für Knoten A:



Hierarchische Addressierung

Eine hierarchische Addressierung ermöglicht kleinere/effizientere Wegwahltabellen.

Beispiel:
Internet-Adressen sind 32-Bit Adressen, die aus zwei Teilen bestehen:

net-id	host-id
--------	---------

übliche Schreibweise: gepunktete Dezimalnotation, z.B.
141.83.21.121

Rechner mit mehreren Netzverbindungen (z.B. Gateways) haben mehrere IP-Adressen

2. Beispiele für adaptive Verfahren

Einträge aus /etc/route.conf, Suse 7.3

192.168.0.0	0.0.0.0	255.255.255.0	eth1
141.89.59.0	0.0.0.0	255.255.255.0	eth0
default	141.89.59.254	0.0.0.0	eth0

Isoierte (lokale) Verfahren von Baran (1964): Jeder Vermittlungsschreiner entscheidet selber aufgrund von selbstgesammelten Informationen.

- Hot-Potato-Routing:** Die empfangene Nachricht wird so schnell wie möglich weitergegeben, d.h. an den Nachbarn mit der kürzesten Warteschlange vor der Ausgangsleitung.

Modifikation: Wegwahltabelle, die nicht nur die kürzesten Wege, sondern auch Alternativwege enthält. Unter diesen wird dann gemäß Warteschlangenlänge entschieden.

- Spalte:** Ziel-Netz
- Spalte:** ausgehende Interface, auf dem das Paket weitergereicht werden soll

2. Backward Learning: Die Pakete zählen unterwegs die Anzahl Vermittlungsrechner, die sie passieren (hopscount). Empfängt ein Rechner A über den Nachbarknoten N ein Paket von Rechner S mit hopscount = k , lernt A , daß es einen Weg der Länge k zu S über N gibt und ergänzt seine Wegwahltabelle:

Ziel	Nachbarknoten	Länge
...
S	N	k

Regeln:

- 1 (Wegwahl) Ist der Empfänger in der Wegwahltabelle eingetragen, wird das Paket an den zuständigen Nachbarknoten geschickt.

Sonst: Fluten

- 2 (Update) Die Wegwahltabelle wird aktualisiert, falls ein Paket eintrifft, das einen kürzeren Weg angibt.

3. Vector Distance Routing

Austausch von Übertragungszeitvektoren:

$$\begin{pmatrix} \tilde{d}(A, Z_1) \\ \tilde{d}(A, Z_2) \\ \dots \\ \dots \\ \dots \\ \tilde{d}(A, Z_n) \end{pmatrix}$$

$\tilde{d}(A, Z_i)$ gibt die aktuelle Schätzung für die Übertragungszeit von A zum Ziel Z_i an (z.B. in Form der Hop-Count-Metrik).

$\tilde{d}(A, Z_i) = \infty$, falls kein Weg bekannt ist.

- Initialisierung der Wegwahltabellen z.B. mittels Dijkstra-Algorithmus oder Backward Learning
- Knoten A mißt periodisch die Laufzeit von Nachrichten zu seinen Nachbarknoten N : $d(A, N)$
- A schickt entweder in regelmäßigen Zeitabständen (synchron) oder nach einer Aktualisierung (asynchron) den Übertragungszeitvektor an alle seine Nachbarn.

$$d(A, Z) := \min\{d_N(A, Z) \mid N \text{ Nachbar von } A\}$$

bestimmt.

- Neuberechnung der Wege: Hat A die Übertragungsvektoren aller seiner Nachbarn erhalten, so wird für jeden Ziellknoten Z und jeden Nachbarn N die geschätzte Weglänge $d(A, Z)$ über N neu berechnet:

$$d_N(A, Z) := d(A, N) + \tilde{d}(N, Z)$$

und

$$d(A, Z) := \min\{d_N(A, Z) \mid N \text{ Nachbar von } A\}$$

9.2 Fragmentierung

Definition 35:

Paketfragmentierung: Das Datenpaket aus Netz 1 wird in mehrere Datenpakete zerlegt, um der Rahmengröße von Netz 2 zu genügen.

Beispiel: Datenfeld im LAN: 1500 - 8274 Byte
Datenfeld in X.25: 126 Byte, 496 Byte

1. Transparente Fragmentierung:

- Jedes Fragment wird an das gleiche „Ausgangsgateway“ adressiert.
Dort werden die Fragmente wieder zusammengesetzt, bevor sie ggf. in das nächste Netz geschickt werden.
- + Netze, in denen das Paket nicht fragmentiert werden muß, durchläuft das Paket als eine Nachricht. \Rightarrow weniger fehleranfällig
 - Pro Netz muß ggf. einmal fragmentiert und reassembliert werden

\Rightarrow bei virtuellen Verbindungen geeignet

- 2. Internetfragmentierung** für Datagrammnetzwerke: Fragmente werden erst am Zielrechner wieder zusammengesetzt.
- + minimaler Reassemblierungsaufwand
 - + flexible Wegwahl: Fragmente können unterschiedliche Ausgangsgateways benutzen
 - Header muß auf alle Fragmente kopiert werden
 - Fehlerkontrolle liegt beim Zielrechner

Kennzeichen:

- Schicht 3 Protokoll,
- Datagram Service,
- Wegwahl, Anpassen der Paketgröße, Adressen etc. beim Übergang in anderes Netz, Flußkontrolle, ...
- **hierarchische Wegwahl:** Routing orientiert sich am Zielnetzwerk, nicht am Zielrechner

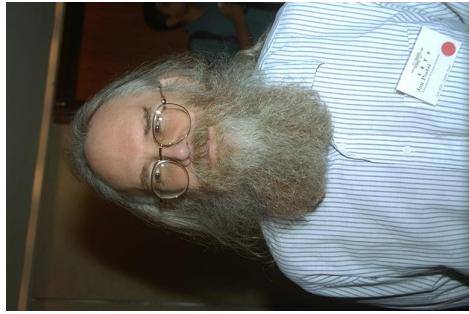
Publikation über TCP/IP:

Vinton Cerf, Robert Kahn: *A Protocol for Packet Network Interconnection*, IEEE Transactions on Communications Technology, Vol. COM-22, No. 5, S. 627-641, 1974.
RFC 791, Jon Postel (1981)
RFC 1122, R. Braden (1989)

Jon Postel (6.8.1943 - 16.10.1998)

Postel began his work on computer networking in the late 1960s, while a graduate student at the University of California at Los Angeles. He was one of a small group of computer scientists who created the Arpanet, the precursor to the Internet. At UCLA in 1969, Postel assisted in the installation of the Arpanet's first communications switch, which routes network traffic.

For nearly 30 years, Postel also served as editor of the Request for Comments series.



Prof. Bettina Schnor

524

Dr. Jonathan B. Postel

Das TCP/IP-Protokoll, die Basis des heutigen Internets, wurde vor den modernen PCs und Workstations entworfen, bevor Ethernet und lokale Netzwerke Verbreitung gefunden hatten, vor dem Web, Audio-/Video-Streaming und Chat.

Cerf und Kahn erkannten die Notwendigkeit eines Netzwerkprotokolls, das einer Vielzahl von noch zu definierenden Anwendungen eine solide Basis bot, und gleichzeitig beliebige Kombinationen von Hosts und Sicherungsschichtprotokollen ermöglichte.

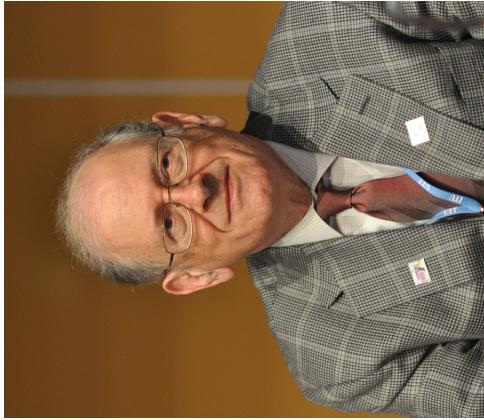
2004 erhielten Cerf und Kahn den ACM Turing Award, den „Nobelpreis der Informatik“, für ihre „Pionierarbeiten im Internetworking, einschließlich des Entwurfs und der Implementation der grundlegenden Kommunikationsprotokolle des Internets, TCP/IP, und für ihre inspirierte Führungssrolle im Bereich Computernetzwerke“.

Vinton Cerf, Robert Kahn und TCP/IP

Quelle: James F. Kurose und Keith W. Ross: Computernetzwerke, S. 273

In den frühen 1970ern wuchs die Zahl der Paketvermittlungsnetze stark an, wobei das ARPAnet - der Vorläufer des Internets - nur eines von vielen Netzen war. Jedes dieser Netze hatte sein eigenes Protokoll. Zwei Forscher, Vinton Cerf und Robert Kahn, erkannten die Wichtigkeit, diese Netze zusammenzuschalten, und entwickelten ein netzwerkübergreifendes Protokoll, das sie TCP/IP (für Transmission Control Protocol/Internet Protocol) nannten.

Während Cerf und Kahn ihr Protokoll zunächst als eine Einheit betrachteten, wurden später die Bestandteile TCP und IP getrennt.



525

Prof. Bettina Schnor

Vinton Cerf (links) und Robert Kahn

Quelle: Wikimedia

Internet-Adressen

Übliche Schreibweise: gepunktete Dezimalnotation, z.B.
141,83 21,121

Internet-Adressen sind 32-Bit Adressen, die aus zwei Teilen bestehen:

net-id	host-id
--------	---------

Vinton Cerf, einer der Väter des Internets, im Vortrag an der Universität Potsdam, 26.5.2011:

"How could we make open standards? It was the time of the cold war!

If you want to blame someone for the 32 bit IPv4 addresses, blame me! – I had to decide, and we expected the network to be experimental!

Private IP-Adressen

Im Gegensatz zu öffentlichen IP-Adressen sind **private IP-Adressen** frei verfügbar, werden aber nicht im Internet weitergeleitet (auch: **nonroutable addresses**).

10/8 10 0000 10 2EE 2EE 2EE

- Vorteil:** Mit privaten IP-Adressen lassen sich Netzbereiche aufbauen, die im Internet nicht sichtbar sind. \Rightarrow freie Designentscheidung, frei verfügbare Adressen
 - Nachteil:** Erfordert *Network Address Translation (NAT)* für Übergang in das Internet.

Übliche Schreibweise: gepunktete Dezimalnotation, z.B.

141.83.21.121

	net-id	host-id
Netzadresse	net-id	0 ... 0
Broadcast-Adresse	net-id	1 ... 1
Limited Broadcast (lokal im LAN)	1 ... 1	1 ... 1
Loopback-Interface	127	beliebig (oft 1)

Rechner mit mehreren Netzverbindungen (z.B. Gateways) haben mehrere IP-Adressen.

Die ICANN (Internet Corporation for Assigned Names and Numbers) ist für das Management im Internet zuständig (u.a. für den Betrieb vom Domain Name System (DNS)).

Adressklassen von IPv4

100

- Class A: (0.0.0.0 - 127.255.255.255),
- Class B: (128.0.0.0 - 191.255.255.255
- Class C: (192.0.0.0 - 223.255.255.255

- Das Konzept der Adressklassen wurde 1993 aufgegeben und durch die Einführung des **Classless Inter-Domain Routing**

(CIDR) ersetzt.
⇒ Durch die Verwendung von variablen Subnetzmasken lässt sich der IP-Adressraum besser ausnutzen.

Prof. Bettina Schnor

Prof. Bettina Schnor

Datagramformat:

0	4	8	16	19	24	31
VERS	HLEN	SERVICE TYPE		TOTAL LENGTH		
IDENTIFICATION			FLAGS	FRAGMENT OFFSET		
TIME TO LIVE	PROTOCOL			HEADER CHECKSUM		
		SOURCE IP ADDRESS				
		DESTINATION IP ADDRESS				
		IP OPTIONS (IF ANY)		PADDING		
			DATA			...

- IP-Adresse und Subnetzmaske werden mittels bitweisem logischen UND verknüpft.
Beispiel: 192.168.1.129 mit Subnetzmaske 255.255.255.0
 - Alternativ: Angabe der Subnetzmaske durch Angabe der gesetzten Bits: /n
Beispiel: 192.168.1.129/24

Prof. Bettina Schnorr

532

33

- **VERS:** Versionsnummer des benutzten IP-Protokolls
 - **PROTOCOL:** Bestimmt das Transportprotokoll, das die Daten erzeugt hat, z.B. welche TCP-/UDP-Version

- Variable Header- und Daten-Lnge
 - optional sind die Felder: **IP OPTIONS, PADDING** \Rightarrow variable Header-Lnge
 - **HLEN**: Header-Lnge in 32-Bit-Worten
 - **TOTAL LENGTH** $< 2^{16} = 65536$ Byte
 - Datenlnge: TOTAL LENGTH - HLEN . 4 [Byte]
 - **HEADER CHECKSUM**: Prfsumme berprft nur den Header:
Einerkomplement von je 16-Bit \rightarrow Einerkomplement der Summe

In jedem Gateway wird die TTL um 1 und zusätzlich für jede Sekunde Wartezeit um 1 runtergezählt.
 $TTL = 0 \Rightarrow$ Datagramm wird vernichtet und der Absender benachrichtigt.

SERVICE TYPE:	Prio	D	T	R	unused
D-Bit:	low delay				
T-Bit:	high throughput				
R-Bit:	high reliability				

Aktuell: Differentiated Services (DS):
CODEPOINT: 6 Bit
UNUSED : 2 Bit

Prof. Bettina Schnor

Internet-Fragmentierung: Fragmente werden erst am Zielrechner wieder zusammengesetzt.

MTU:= maximum transfer unit:= maximale Länge des Datenfeldes im Rahmen

VERS	4	8	31	1 6	1 9	2 4	TOTAL LENGTH
IDENTIFICATION				FLAGS	FRAGMENT OFFSET		
TIME TO LIVE	PROTOCOL	SOURCE IP ADDRESS	DESTINATION IP ADDRESS	IP OPTIONS (IF ANY)	PADDING	DATA	...

Prof. Bettina Schnor

536

Prof. Bettina Schnor
537

IP Forward-Algorithm: RouteDatagram (Datagram, Wegwahltabelle)

- 1.) Bestimme die Ziel-IP-Adresse des Datagramms D und den zugehörigen Netzwerk-Präfix N
- 2.) **IF** N ist ein direkt verbundenes Netzwerk
Then schicke das Datagramm auf dem zugehörigen Interface raus
ELSE IF Wegwahltabelle besitzt einen Eintrag für Ziel-IP
THEN schicke Datagramm zum nächsten Router gemäß Eintrag
ELSE IF Wegwahltabelle besitzt einen Eintrag für Netzwerk N
THEN schicke das Datagramm zum nächsten Router gemäß Eintrag
IF Wegwahltabelle enthält Default-Eintrag
THEN schicke das Datagramm zum Default-Router gemäß Eintrag
ELSE Routing-Error!

Prof. Bettina Schnor

538

Prof. Bettina Schnor
539

Address Resolution Protokoll (ARP)

IP Packet Processing - Outgoing Packet

- RFC 826, 1982, David C. Plummer: An Ethernet Address Resolution Protocol - or - Converting Network Protocol Addresses to 48 Bit Ethernet Address for Transmission on Ethernet Hardware
- IP-Adresse \mapsto MAC-Adresse von Schicht 2
- ARP merkt sich seine gelernten Einträge eine gewisse Zeit lang im **ARP Cache**

Wie lernt man Einträge?

- 1 **ARP Request:** Maschine A schickt einen Broadcast „Wer hat IP-Adresse I_B ?“

Der ARP Request enthält die MAC Adresse M_A und die zugehörige IP-Adresse I_A von A.

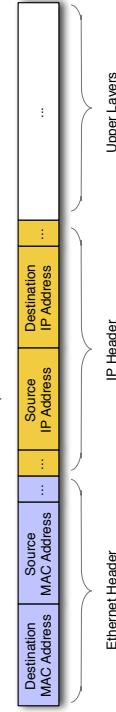
- 2 Alle Empfänger aktualisieren ihren ARP Cache mit (I_A, M_A) .
- 3 Maschine B antwortet mit (I_B, M_B) .

Prof. Bettina Schnor

Step 2: Fill in Source IP-Address

- If a host has only one configured interface/address, the IP-Address of this interface is used.
- If a host has multiple addresses the kernel performs a Route-Lookup over the routing table to determine the closest matching route to the destination. The source of this route becomes the Source IP-Address.

Step 2:
Route-Lookup
to find matching
IP-Address
of Interface



Prof. Bettina Schnor

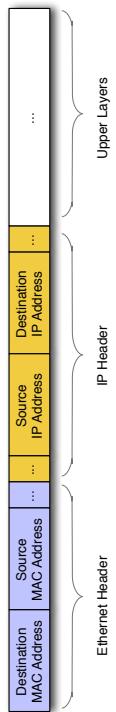
Step 3: Fill in Source HW-Address

- The Hardware(MAC)-Address of the Source Interface is used.

Step 4: Fill in Destination HW-Address

- The Network Address of the IP-Destination is compared with the local IP-Address (Subnet-Mask) in order to determine if the Destination is local or remote.

Step 3:
HW-Address
of Interface



Prof. Bettina Schnor

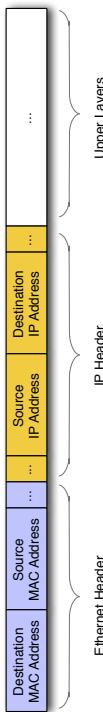
540 Prof. Bettina Schnor
541 Prof. Bettina Schnor

What happens when the Kernel sends an IP-Packet?

Step 1: Fill in Destination IP-Address

- The application supplies the Destination IP-Address (numeric Address or Hostname)
- This usually requires a DNS-Lookup, in order to map a Hostname to an IP-Address

Step 1:
IP-Destination
Address
(e.g. from DNS)



541 Prof. Bettina Schnor

543 Prof. Bettina Schnor

Step 4: Fill in Destination IP-Address (continued...)

Local Destination: Packet can be send directly to the Host -> Destination MAC: Host

Remote Destination: Packet is send to the Next-Hop-Router -> Destination MAC: Router

What can go wrong?

- Wrong Subnet Mask → Leads to wrong L2-Next Hop decision.
- Wrong Gateway Address → Packets are not forwarded correctly.
- Outdated ARP-Cache entry → Packets are not forwarded correctly.
- No entry in the Routing table → Packet will be dropped.
- Wrong entries in the Host routing table → Source IP-Address and Next Hop Information may be wrong.
- ...
- The ARP-Cache is consulted to find the MAC-Address of the L2-Destination Host or Router
- If a corresponding entry is found, it is used as Destination MAC-Address.
- Otherwise an ARP-Request is send as a Broadcast to the local LAN asking the holder of the IP-Address to send back their MAC-Address (IPv6 uses ICMP-Neighbor Detection instead of ARP).

Einträge aus /etc/route.conf, Suse 7.3

192.168.0.0	0.0.0.0	255.255.255.0	eth1
141.89.59.0	0.0.0.0	255.255.255.0	eth0
default	141.89.59.254	0.0.0.0	eth0

IP Wegwahltabellen

Definition 36:

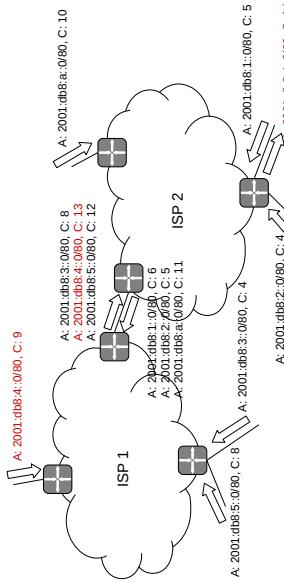
- 1 Eine Menge von Netzen und Routern, die administrativ zusammengehören, heißen **Autonomes System**. Mittels des **Exterior Gateway Protocol (EGP)** findet Wegwahl zwischen Autonomen Systemen statt.
- 2 Wege innerhalb eines autonomen Systems werden mit dem **Interior Gateway Protokoll (IGP)** bestimmt.

- Beispiel für ein Autonomes System: DFN ist AS680
- Beispiel für ein Exterior Gateway Protokoll: **Border Gateway Protocol (BGP)**
1. Spalte: Ziel-Netz
2. Spalte: IP vom nächsten Router/Gateway,
Adresse 0.0.0.0 bedeutet, daß der Zielrechner direkt
angeschlossen ist (im LAN)
3. Spalte: Netzmaske
4. Spalte: ausgehende Interface, auf dem das Paket weitergereicht
werden soll

Tobias Tragsdorf: „Sicheres BGP Routing“, Diplomarbeit,
Universität Potsdam, 2012.

Border Gateway Protocol

Beispiele für IGP-Protokolle:



1 80er Jahre: RIP (Routing Information Protocol)

RFC 1058 (1988), RFC 2453 (1998)

weit verbreitet: als routed im 4 BSD-UNIX basiert auf Vector Distance Routing:
Austausch der Vektoren alle 30 s, $f = 1$ (Hop-Count-Metrik)

A: 2001:db8:4::/60 C: 9
A: 2001:db8:4::/60 C: 13
A: 2001:db8:5::/60, C: 12

A: 2001:db8:3::/60, C: 8
A: 2001:db8:5::/60, C: 10

A: 2001:db8:2::/60, C: 5
A: 2001:db8:a::/60, C: 11

A: 2001:db8:1::/60, C: 6
A: 2001:db8:a::/60, C: 11

A: 2001:db8:2::/60, C: 4
A: 2001:db8:1::/60, C: 5
A: 2001:db8:4::/60 C: 14

■ Y. Rekhter, T. Li, S. Hares: *A Border Gateway Protocol 4 (BGP-4)*, RFC 4271, 2006

■ Ziel: Routing über Netzgrenzen hinweg (OSI-Layer: 3)

■ Benachbarte Router bauen Verbindung über TCP-Port 179 auf

■ Annoncierung von Routeninformationen

■ (ökonomische) Routingentscheidungen

Prof. Bettina Schnor

548

Prof. Bettina Schnor
549

Beispiel: Welche Autonomen Systeme passiert mein Nachrichtenverkehr?

> /sbin/traceroute -a
> /sbin/traceroute -an
> /sbin/tracepath

Internet Control Message Protocol (ICMP)

ICMP legt Regeln zur Kommunikation zwischen verschiedenen IP-Instanzen fest zum Austausch von

- Fehlernachrichten (z.B. destination unreachable, route change request)
- Statusmeldungen (z.B. echo request)

ICMP ist Teil von IP. ICMP-Nachrichten werden als IP-Nachricht verschickt (**Tunneling**).
Wie erkennt IP, dass es sich um ein ICMP-Paket handelt?

protocol = 1 \Rightarrow ICMP-Nachricht

ICMP Type Field	Message
0	Echo Reply
3	Destination Unreachable
4	Source Quench
5	Redirect (change a route)
8	Echo Request
9	Router Advertisement
10	Router Solicitation
11	Time Exceeded for a Datagram
12	Parameter Problem on a Datagram
13	Timestamp Request
14	Timestamp Reply
15	Information Request (obsolete)
16	Information Reply (obsolete)
17	Address Mask Request
18	Address Mask Reply

Beispiel:
ping: Kommandozeilen-Tool, das eine Folge von ICMP-Nachrichten vom Typ **Echo Request** schickt und ICMP **Echo Reply** Nachrichten als Antwort bekommt.

⇒ Verschiedene ICMP-Nachrichten werden über das **ICMP Type** Feld spezifiziert (weitere Informationen können im **ICMP CODE** Feld stehen).

ICMP Type Field	Message
0	Echo Reply
3	Destination Unreachable
4	Source Quench
5	Redirect (change a route)
8	Echo Request
9	Router Advertisement
10	Router Solicitation
11	Time Exceeded for a Datagram
12	Parameter Problem on a Datagram
13	Timestamp Request
14	Timestamp Reply
15	Information Request (obsolete)
16	Information Reply (obsolete)
17	Address Mask Request
18	Address Mask Reply

Prof. Bettina Schnor 552

Beispiel: Echo Request und Echo Reply Format

0	TYPE (8 or 0)	8	16
31	IDENTIFIER	CODE = 0	CHECKSUM
		SEQUENCE NUMBER	OPTIONAL DATA
			...

Prof. Bettina Schnor 553

Das ICMP Type Feld definiert

- 1 Bedeutung der ICMP-Nachricht
- 2 Format der ICMP-Nachricht

ICMP-Header:

0	8	16
Type Field	Code	Checksum

CHECKSUM: wird analog wie bei IP gebildet

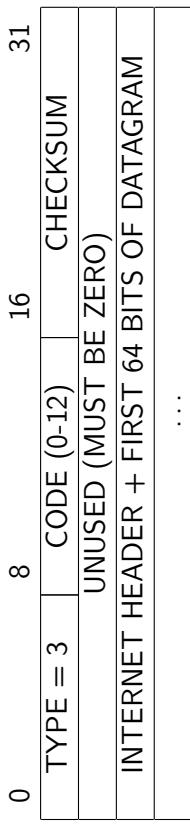
SEQUENCE NUMBER: benutzt Sender um Anfragen und Antworten einander zuordnen zu können
 OPTIONAL DATA: In der Reply-Nachricht werden exakt die Daten aus der Request-Nachricht wiederholt (variable Länge)

Prof. Bettina Schnor 554

Prof. Bettina Schnor 555

Code Value	Meaning
0	Network unreachable
1	Host unreachable
2	Protocol unreachable
3	Port unreachable
4	Fragmentation needed and DF set
5	Source route failed
6	Destination network unknown
7	Destination host unknown
8	Source host isolated
9	Communication with destination network administratively prohibited
10	Communication with destination host administratively prohibited
11	Network unreachable for type of service
12	Host unreachable for type of service

Beispiel: ICMP-Nachricht **Destination Unreachable**
 Wenn ein Router ein IP Datagram nicht weiterrouten kann, schickt er eine **Destination Unreachable**-Nachricht an den Absender zurück.



CODE: beschreibt das Problem genauer

556	Prof. Bettina Schnor
557	Prof. Bettina Schnor

Beispiel:

traceroute - print the route packets trace to network host
 traceroute6

8. Der IEEE-Standard 802 für lokale Netze

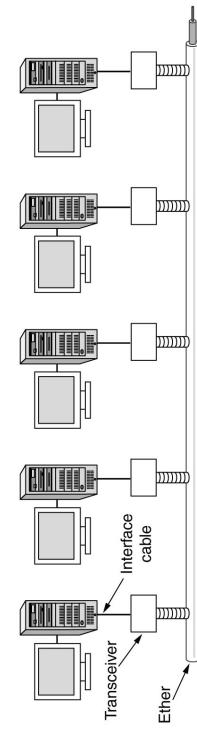
	Logical Link Control (LLC)		
	802.2		
1	Medium Access Control Protocols (MAC)	Tokenring	
2	CSMA/CD	Tokenbus	
3	802.3	802.4	802.5
4	Physical Layer		

- 1 Ethernet (IEEE 802.3): CSMA/CD
- 2 Digital Subscriber Line (DSL)
- 3 PPP over Ethernet (PPPoE)
- 4 WLAN-Technologie mittels Ethernet

Literatur: Rich Seifert: *Gigabit Ethernet: Technology and Applications for High-Speed LANs*, Addison-Wesley, 1998

8.1 Ethernet

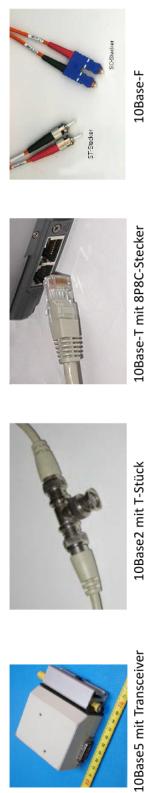
- von Xerox in den 70er Jahren entwickelter und als IEEE 802.3
- 1983 standardisierter serieller Bus,
- "luminiferous ether" (leuchtender Äther),



- 10 MBit/s Übertragungsrate,
- bis zu 1024 Stationen über maximal 2,5 km,
- Koaxialkabel, Twisted Pair, Glasfaser
- Manchesterkodierung
- basiert auf 1-persistent CSMA
- 46 Bytes ≤ Paketgröße ≤ 1500 Bytes
- aktuell: FastEthernet 100 MBit/s, Gigabit Ethernet, 10 Gigabit

Kabeltypen nach 802.3

Bezeichnung	Kabel	max Segmentlänge	Knoten/ Segment	Vorteile
10Base5	Dickes Koax	500m	100	gut für Backbones
10Base2	Dünnes Koax	200m	30	kostengünstig
10Base-T	Verdrilltes Paar	100m	1024	einfache Wartung
10Base-F	Glasfaser	2000m	1024	ideal zwischen Gebäuden



Quelle: Schoelzel

Ethernet Paketformat (1983-1996)

Präambel	Start-begrenzer	1	6	6	Quell-adresse	Länge des Datenfeldes	2	0-1500 Daten	Pad	0-46 Prüfsumme	4
----------	-----------------	---	---	---	---------------	-----------------------	---	--------------	-----	----------------	---

Minddestänge von Zieladresse bis zur Prüfsumme: 64 Bytes \Rightarrow
Padding

Prüfsumme: CRC-32

\Rightarrow Random-Zugriffsverfahren

CSMA/CD-Verfahren (Carrier Sense Multiple Access with Collision Detection)

- Die sendewillige Station hört das Kabel ab und sendet, falls das Kabel frei ist (**Carrier Sense**).
 - Andernfalls wartet die Station, bis das Kabel frei wird (1-persistent).
- Da alle sendewilligen Stationen gleichberechtigt sind (**Multiple Access**), kann es zu Kollisionen kommen. Jeder Sender hört deshalb auch die eigene Nachricht mit und überprüft, ob sie ungestört bleibt (**Collision Detection**).
 - Im Fehlerfall wird ein Jamming-Signal geschickt und der Sendevorgang nach einem zufällig gewählten Zeitintervall wiederholt (**Binary Exponential Backoff Algorithm**).

Eigenschaften CSMA/CD

- Random-Zugriffsverfahren**, daher nicht für Echtzeitanforderungen geeignet
- Ethernet spezifiziert eine Bitfehlerrate (Bit Error Rate (BER)) von 10^{-8} im Worst-Case für Kupfer.
Bei Sprachübertragung ist eine BER von 10^{-6} und kleiner hinreichend.
- Gigabit Ethernet spezifiziert eine BER von 10^{-10} , und 10 Gigabit Ethernet von 10^{-12} .
 - Pakete können wegen Überlast im Switch oder von der Karte weggeworfen werden. Dies wird vom Protokoll nicht bemerkt und folgerichtig auch nicht behoben. \Rightarrow **Keine Flusskontrolle**.
 - FIFO-Reihenfolge der Pakete zwischen zwei Kommunikationspartnern wird eingehalten.

CSMA/CD-Verfahren (Carrier Sense Multiple Access with Collision Detection)

474

Prof. Bettina Schnor

CSMA/CD-Verfahren (Carrier Sense Multiple Access with Collision Detection)

475

Prof. Bettina Schnor

Binary Exponential Backoff Algorithm

- Zeit wird in "Slots" eingeteilt: slot = $2 t_p$
- nach 1. Kollision: 0 oder 1 Slot warten
- nach 2. Kollision: 0, 1, 2 oder 3 Slots warten
- allg.: nach i-ter Kollision: 0, 1, ..., oder $2^i - 1$ Slots warten
- ab 10. Kollision: 0, 1, ..., oder 1023 Slots warten
- Abbruch nach 16. Kollision und Fehlermeldung an nächsthöhere Schicht

Neuere Entwicklungen

- heutzutage üblich: Sterntopologie via Switch
 ⇒ CSMA/CD wird nicht eingesetzt
- Moderne Netzwerkkarten und Switche erlauben **Jumbo-Frames** von z.B. 9000 Bytes,
- **MAC Control Protocol:** Einige Karten und Switche implementieren das in IEEE 802.3x standardisierte Verfahren zur Flusskontrolle: Erreicht der Empfangspuffer der Netzwerkkarte einen kritischen Wert, schickt die Karte einen PAUSE-Paket an den Sender mit der Zeitangabe, wie lange der Sender die Paketversendung stoppen soll.
- Änderung im Paketformat: Neues Type-Feld zum schnellen Erkennen von MAC Control Frames (IEEE 802.3x, 1997), wird von Gigabit Ethernet benutzt.



Bem.: Hidden Station-Problem in **drahtlosen** Netzen:

Prof. Bettina Schnor

478

Viele Protokolle für **drahtlose** Kommunikation wie z.B. WLAN arbeiten gemäß des **CSMA/CA-Verfahrens (Carrier Sense Multiple Access with Collision Avoidance)**:

- Die sendewillige Station hört das Medium ab und würfelt, falls das Medium für *DIFS* Zeiteinheiten frei ist, eine *Backoffzeit*. Nach Ablauf der Backoffzeit wird gesendet.
- Ist das Medium belegt, wartet die Station bis der Network Allocation Vector (NAV) auf Null ist, und wiederholt den ersten Schritt.
Empfängt eine Station eine Nachricht mit der Information, wie lange das Medium belegt sein wird, merkt sich die Station diesen Wert im NAV und zählt ihn runter.

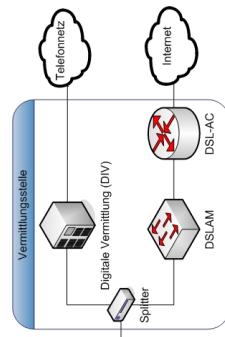
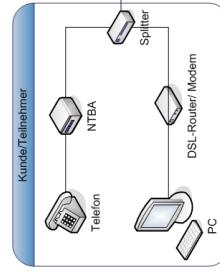
Prof. Bettina Schnor

479

8.2 Digital Subscriber Line (DSL)

Breitband-Internetzugang über Kupfer

In DSL-Netzen sammeln sogenannte **DSL-Access Multiplexer (DSLAMs)** den Datenverkehr der vor Ort angeschlossenen Teilnehmer und leiten ihn an einen Router, den sogenannten **DSL Access Concentrator (DSL-AC)** weiter. Der DSL-AC ist also der erste Hop auf dem Weg in das Internet.



(Quelle: Wikipedia)

DSL-Access Multiplexer (DSLAM):



(Quelle: Wikipedia)

Telekom-DSL nutzt: PPPoE

PPPoE Paketformat

Klassisch Ethernet:

7	1	6	6	2	0-1500	0-46	4
Präambel	Start-begr.	Ziel-adr.	Quell-adr.	Länge/Typ	Daten	Pad	Prüfsumme

PPPoE Paketformat:

7	1	6	6	2	6	4	
Präambel	Start-begr.	Ziel-adr.	Quell-adr.	Type = 0x8863 bzw. 0x8864	PPPoE-Paket-Header	Daten oder Padding	Prüfsumme

Interpretation des Länge/Typ-Feldes:

The ETHER_TYPE is set to either 0x8863 (Discovery Stage) or 0x8864 (PPP Session Stage).

Discovery-Stage

1 Der Host/Client schickt ein **Initiation Packet** an die Broadcast-Adresse.

2 Einer oder mehrere Access Concentrators antworten mit **Offer Packet**.

3 Der Host/Client wählt einen Access Concentrator aus.
“The choice can be based on the AC-Name or the Services offered.”

4 Der Host/Client schickt ein **Unicast Session Request Packet** an den ausgewählten Access Concentrator mit SESSION_ID=0x0000.

5 Der Access Concentrator erzeugt für diese Session eine eindeutige Session-ID und schickt diese mit einem **Confirmation Packet** an den Host/Client.

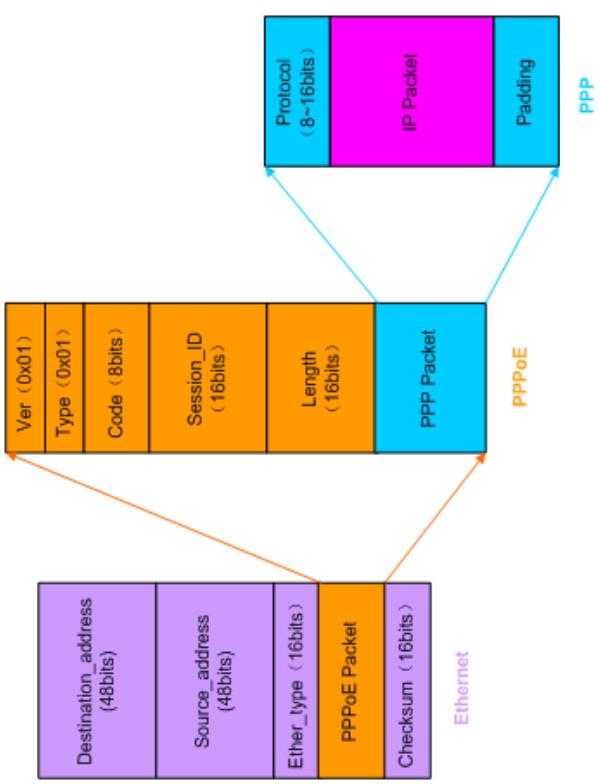
- RFC 2516, Anno 1999:
“In many access technologies, the most cost effective method to attach multiple hosts to the customer premise access device, is via Ethernet.”
- PPPoE besteht aus einer **Discovery-** und einer **Session-Phase**.
Während der Discovery-Phase wird die Ethernet-Adresse des Servers gelernt und eine eindeutige Session-ID ausgehandelt:
“In the Discovery process, a Host (the client) discovers an Access Concentrator (the server). Based on the network topology, there may be more than one Access Concentrator that the Host can communicate with. The Discovery stage allows the Host to discover all Access Concentrators and then select one.”

Potsdamer Variante von Art-efx

(Quelle: Wikipedia)

Telekom-DSL nutzt: PPPoE

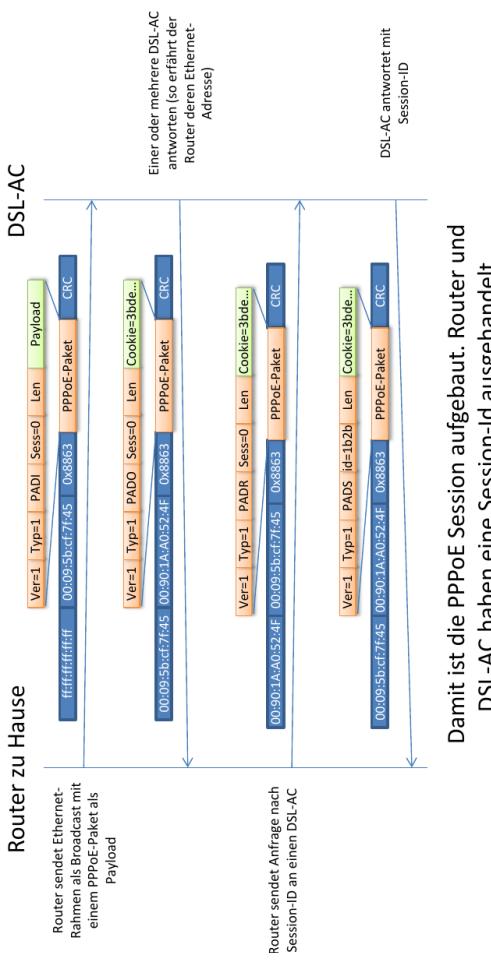
Discovery Stage: The ETHER_TYPE is set to 0x8863



Quelle: <http://www.h3c.com>

Prof. Bettina Schnor

486



Quelle: <http://www.mwlab.net/art/pppoe/>

487

Session-Stage

Sind Ethernet-Adresse und Session-ID ausgetauscht, beginnt die Session-Phase. Ab jetzt werden PPP-Pakete in der Ethernet-Payload verschickt (**PPP Encapsulation**).

Besonderheiten zu LCP

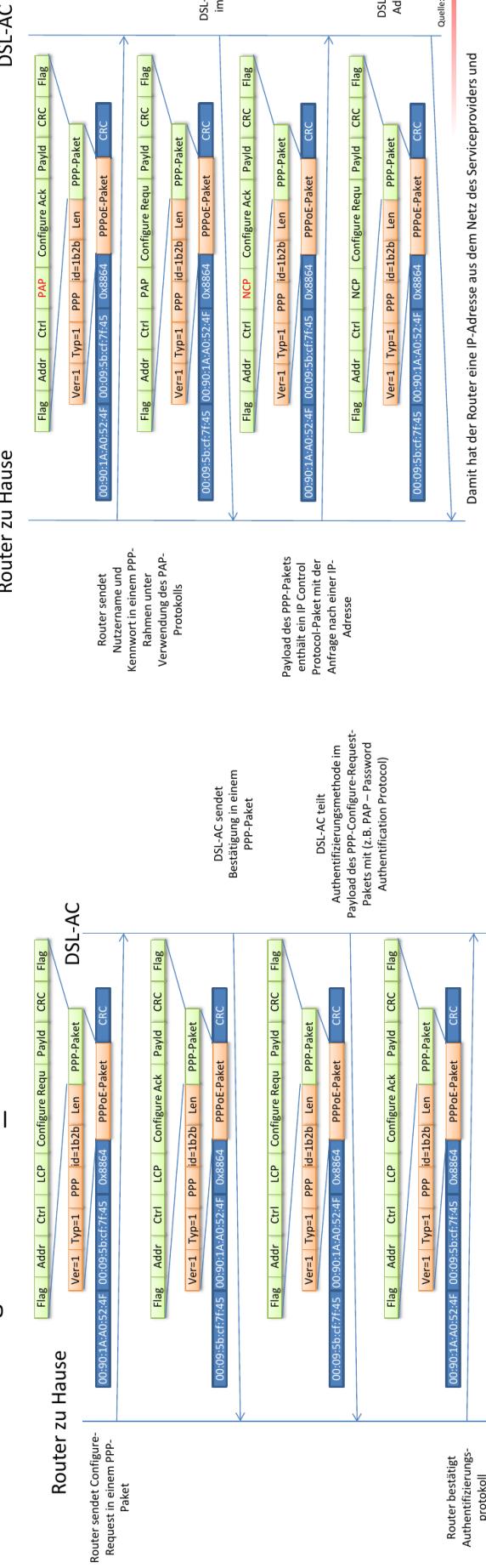
- An implementation MUST NOT request any of the following options, and MUST reject a request for such an option:
 - Field Check Sequence (FCS) Alternatives,
 - Address-and-Control-Field-Compression (ACFC),
 - Asynchronous-Control-Character-Map (ACCM)
- The Maximum-Receive-Unit (MRU) option MUST NOT be negotiated to a larger size than 1492. Since Ethernet has a maximum payload size of 1500 octets, the PPPoE header is 6 octets and the PPP Protocol ID is 2 octets, the PPP MTU MUST NOT be greater than 1492.

487

Router sendet Ethernet-Rahmen als Broadcast mit einem PPPoE-Paket als Payload

- Einer oder mehrere DSL-AC antworten (so erfaßt der Router deren Ethernet-Adresse)
- DSL-AC antwortet mit Session-ID

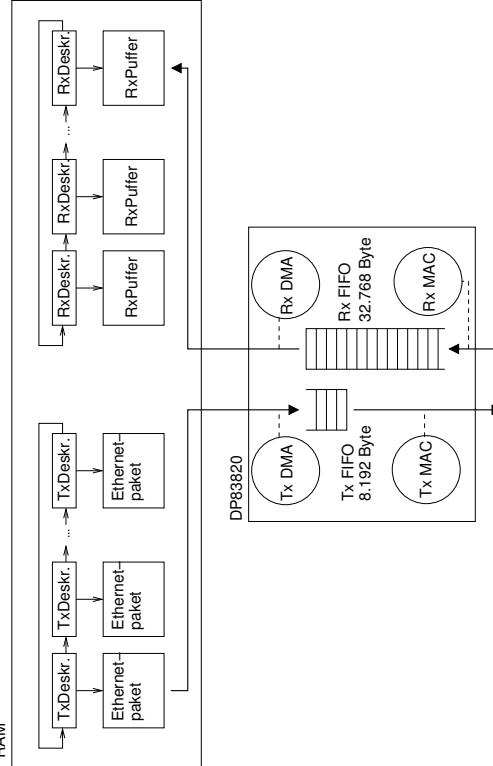
Session Stage: The ETHER TYPE is set to 0x8864



Prof Bettina Schnorr

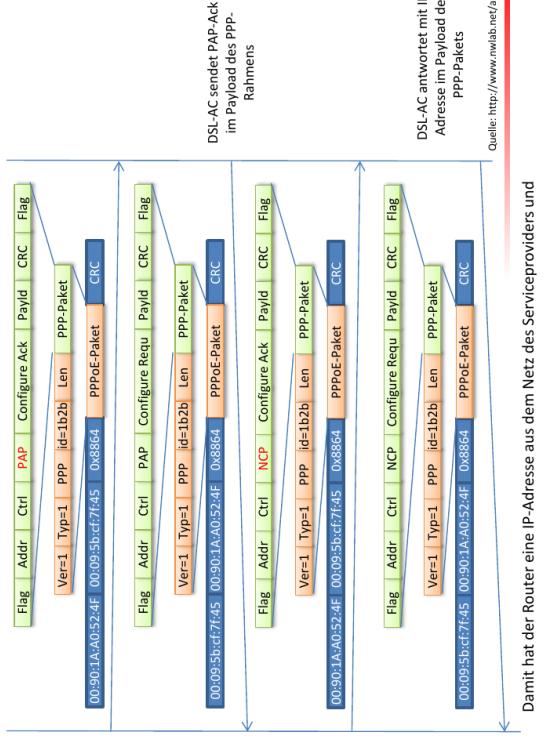
FIFO-Puffer im Contoller DP83820 auf der Gigabit Ethernet Karte GA 621 und Descriptorringe im Hauptspeicher

11



Quelle: Marco Ehler, Diplomarbeit

Router zu Hause



Quelle: <http://www.nwlab.net/art/ppoe/>

Prof Bettina Schnor

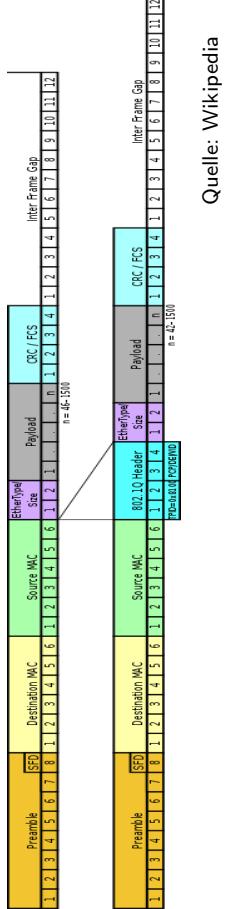
8.4 WLAN-Technologie mittels Ethernet

11

- Das lokale Netz wird in *virtuelle Netze* aufgeteilt. Ziel ist es Netzwerksegmente zu separieren:
 - Beispiel: Die Rechner sekretariat1, sekretariat2, sekretariat3 sollen in VLAN 1 liegen.
Die Rechner labor1, labor2, ..., labor42 liegen in VLAN 2.
 - Separationsregel: Pakete von VLAN 1 dürfen nicht in VLAN 2 und umgekehrt.
 - IEEE 802.1Q unterstützt **virtuelle LANs (VLANs)** in einem 802.3 Ethernet. Switches müssen diese Funktionalität ebenfalls unterstützen.

802.1Q Tag Format

- Tag protocol identifier (TPID) (16 Bit):
 - An der Position, an der sonst das EtherType/Länge-Feld steht.
Wird auf TPID = 0x8100 gesetzt, was bedeutet, daß es sich um einen Ethernetrahmen gemäß 802.1Q handelt.
 - Das eigentliche EtherType/Länge-Feld wird um 4 Bytes verschoben und die Länge der Payload um 4 Byte reduziert.
 - Byte 3 und 4 sind Tag control information (TCI) und werden in PCP, DEI und VID unterteilt.



Quelle: Wikipedia

802.1Q definiert ein neues VLAN-Tag im Ethernet-Header:

THERE'S BEEN A LOT OF CONFUSION OVER 1024 vs 1000,
KBYTE vs KBIT, AND THE CAPITALIZATION FOR EACH.
HERE, AT LAST, IS A SINGLE, DEFINITIVE STANDARD:

SYMBOL	NAME	SIZE	NOTES
kB	KILOBYTE	1024 BYTES OR 1000 BYTES	1000 BYTES DURING LEAP YEARS, 1024 OTHERWISE
KB	KELLY-BOOTLE STANDARD UNIT	1012 BYTES	COMPROMISE BETWEEN 1000 AND 1024 BYTES
kiB	IMAGINARY KILOBYTE	1024.51 BYTES	USED IN QUANTUM COMPUTING
kb	INTEL KILOBYTE	1023.937528 BYTES	CALCULATED ON PENTIUM FPU.
Kb	DRIVEMAKERS KILOBYTE	CURRENTLY 908 BYTES	SHRINKS BY 4 BYTES EACH YEAR FOR MARKETING REASONS
KBa	BAKER'S KILOBYTE	1152 BYTES	9 BITS TO THE BYTE SINCE YOU'RE SUCH A GOOD CUSTOMER

Quelle: www.xkcd.com/394/

Prof Bettina Schnor

Prof. Bettina Schnorr 405

Einheiten im Netzwerkbereich

Einheiten im Netzwerkbereich

- 1 Wie gibt man Bandbreiten für Netzwerke an?
 - 1 MB/s oder 1 MBit/s? \Rightarrow Angabe in MBit/s
 - Beispiel: Fast Ethernet mit theoretischer Bandbreite 100-MBit/s
 - 1 MBit/s = 1 000 000 Bit/s, oder
 - $1 \text{ MBit/s} = 1 \cdot 2^{20} = 1\,048\,576 \text{ Bytes/s}$
 - Netzwerkbereich: Basis 10 gemäß dem SI-Standard**
 - \Rightarrow **Beispiel: Fast Ethernet 100-MBit/s = 100 000 000 Bit/s**

Prof. Bettina Schnor

IEC 60027-2

Standardisierung von Einheiten

1. Le Systeme international d'unités (SI) benutzt Basis 10
2. International Electrotechnical Commission (IEC) spezifiziert in IEC 60027-2 2005, third Edition. Einheiten zur Basis 2
kibibytes, mebibytes, gibibytes, exbibytes, 2000

Faktor	Name	Symbol
2^{10}	kibi	Ki
2^{20}	mebi	Mi
2^{30}	gibi	Gi
2^{40}	tebi	Ti
2^{50}	pebi	Pi
2^{60}	exbi	Ei

7. Sicherungsschicht

7.1 Aufgaben der Sicherungsschicht

- 1 Aufgaben der Sicherungsschicht
- 2 Fehlererkennung und Fehlerkorrektur
- 3 Flusssteuerung
- 4 High-level Data Link Control (HDLC)
- 5 PPP – Point-to-Point-Protocol

a) mittels Anfangs- und Endflags (Begrenzer):

Rahmenformat:

SD	Steuerinformation von Schicht 2	Daten	ED
----	---------------------------------	-------	----

SD := Startdelimiter, ED := Enddelimiter

Beispiel: SD = ED = 0 11 11 11 0

Diese Zeichenkette darf in den Daten **nicht** vorkommen.

⇒ **Bitstopfen** := nach 5 aufeinander folgenden Einsen fügt der Sender eine 0 ein. Empfänger streicht nach 5 aufeinander folgenden Einsen eine Null heraus.

Beispiel:

Daten:

Übertragen wird: SD 011011111010111111001 ED

Beispiel: HDLC

7.1.2. Fehlerkontrolle

- b) Mischverfahren: SD + Feld mit Längenangabe
⇒ Schicht 2 muß Anzahl Bits zählen
Beispiel: Ethernet
- c) Verstöße gegen Kodierungsregeln
Beispiel: Tokenbus

Link-to-Link-Überprüfung:

Wurde der Rahmen korrekt übertragen?

Ansatz: Sender berechnet und schickt Prüfsumme (Redundanzbits).

Empfänger berechnet ebenfalls Prüfsumme für den empfangenen Rahmen. „Stimmt“ die Prüfsumme, dann ist die Übertragung („höchstwahrscheinlich“) korrekt und die Daten werden an Schicht 3 weitergegeben. Andernfalls wird der Rahmen weggeworfen.

Beispiel: Hamming-Code, Cyclic Redundancy Code (CRC)

7.1.3. Flußsteuerung

Problem:

Unterschiedlich schnell arbeitender Sender und Empfänger.
Falls der Empfänger langsammer arbeitet als der Sender, können beim Empfänger Datenpuffer überlaufen und Rahmen verloren gehen.

\Rightarrow Protokolle zur Synchronisation des Datenflusses zwischen Sender und Empfänger,
z.B. **Stop-and-wait-Protokolle, Schiebefenster-Protokolle**

Ansatz: **Codewort** der Länge $m + r$

m	Datenbits	r	Redundanzbits
---	-----------	---	---------------

Fehlerkorrekturcodes

$$\begin{array}{l} 2^m \text{ gültige Wörter} \\ b_1 \dots b_m \xrightarrow{\text{Codierung}} b_1 \dots b_m \\ \text{Fehler: } b_1 \dots \overline{b_i} \dots b_m \end{array}$$

Definition 32:

Der **Hamming-Abstand** $h(b_1 \dots b_m, b'_1 \dots b'_m)$ ist die Anzahl der unterschiedlichen Bits, d.h. die Anzahl i mit $b_i \neq b'_i$.

Prof. Bettina Schnor
439

Definition 33:

- 1 Ein Übertragungsfehler heißt **l-Bit-Fehler**, falls $h(b_1 \dots b_m, r(b_1 \dots b_m)) = l$ gilt, wobei $r(b_1 \dots b_m)$ das empfangene Wort bezeichne.

Theorem 4: Gegeben sei eine Codierung C
 $C(b_1 \dots b_m) = b_1 \dots b_m \quad b_{m+1} \dots b_{m+r}$

- a) Ist der minimale Hammingabstand aller gültigen Codewörter $d + 1$ so lassen sich alle l -Bit-Fehler, $l = 1, \dots, d$ erkennen.
 b) Der minimale Hammingabstand aller Codewörter ist $2d + 1$. \Leftrightarrow Es lassen sich alle l -Bit-Fehler, $l = 1, \dots, d$ erkennen und korrigieren.

Prof. Bettina Schnor
440

Beispiel für 1-Bit-Korrekturcode (Hamming 1950)

Das Datenwort wird aufgefüllt mit Redundanzbits (**Checkbits**) an den Positionen b_i , $i = 2^j$, $j = 0, 1, 2, \dots$. Checkbits sind **Parity Bits**:

Bit an Position k wird von allen Checkbits b_{2^j} geprüft, für die gilt, daß 2^j in der Dualzahldarstellung von k vorkommt.
Beim Empfangen werden alle Checkbits überprüft. Falls Checkbit b_i falsch, setze counter := counter + i
Bit b_k verfälscht \Rightarrow alle Checkbits b_i , die b_k mitprüfen sind verfälscht

$$\begin{array}{lcl} \text{counter} = 0 & \Leftrightarrow & \text{Codewort ist korrekt empfangen} \\ \text{counter} = k & \Leftrightarrow & \text{Bit } b_k \text{ ist verfälscht} \end{array}$$

Fehlererkennung mittels Cyclic Redundancy Code (CRC)

Bemerkung:

- 1 Mit dem Hamming-Code können 1-Bit-Fehler erkannt und korrigiert werden.
- 2 Wird im IEEE 802.16 Netzen (drahtlose Breitbandnetze, Wireless Man) benutzt. Bekannt unter dem Namen WiMAX (Worldwide Interoperability for Microwave Access)
Statt WiMAX hat sich mittlerweile LTE durchgesetzt.

zyklische Blocksicherung, Polynomcodes

$$\begin{array}{lcl} b_1 \dots b_m & \longmapsto & b_1 x^{m-1} + b_2 x^{m-2} + \dots + b_{m-1} x^1 + b_m \\ 110001 & \longmapsto & x^5 + x^4 + \dots + 1 = : M(x) \end{array}$$

gegeben: „Generatorpolynom“ $G(x)$

Idee: Das zum Codewort zugehörige Polynom soll in $\mathbb{Z}_2[x]$ durch $G(x)$ teilbar sein.

Fehlererkennung: Das empfangene Wort wird auf das zugehörige Polynom abgebildet.

Geht Polynomdivision in $\mathbb{Z}_2[x]$ mit $G(x)$ auf?

$$\left\{ \begin{array}{llll} \text{Ja} & \hat{=} & \text{höchstwahrscheinlich} & \text{kein Fehler} \\ \text{Nein} & \hat{=} & \text{Fehler!} & \end{array} \right.$$

Prof. Bettina Schnor

443

444

Algorithmus zur Berechnung des Codeworts

$$\text{Codewort } b_1 \dots b_m \quad \underbrace{b_{m+1} \dots b_{m+r}}_{\text{Checksumme}}$$

- 1 $r := \text{Grad}(G(x))$

- 2 Polynomdivision mit Rest:
Bestimme $R(x)$ mit

$$x^r \cdot M(x) = h(x) \cdot G(x) + R(x) \text{ in } \mathbb{Z}_2[x]$$

und $\text{Grad}(R(x)) \leq r$

- 3 Bestimme die zum Polynom

$$T(x) = x^r \cdot M(x) - R(x) = x^r \cdot M(x) + R(x)$$

gehörige Bitkette $b_1 \dots b_m \quad b_{m+1} \dots b_{m+r}$
Checksum $\hat{=} \text{Rest}$

Bemerkung:

- 1 Addition, Subtraktion in $\mathbb{Z}_2 \hat{=} \text{Exklusiv Oder}$

- 2 Übertragungsfehler:

$$\begin{aligned} \text{Empfangen wird } T(x) + E(x) \\ (T(x) + E(x)) / G(x) = h(x) + \frac{E(x)}{G(x)} \\ G(x) \mid E(x) \iff \text{Fehler wird nicht entdeckt} \end{aligned}$$

- 3 $(x+1) \mid G(x) \implies$ Beliebige ungerade Anzahl von Bit-Fehlern wird erkannt.

- 4 Beispiele für Generatorpolynome:

$CRC - 12$	$= x^{12} + x^{11} + x^3 + x^2 + x + 1$
$CRC - 16$	$= x^{16} + x^{15} + x^2 + 1$
$CRC - CCITT$	$= x^{16} + x^{12} + x^5 + 1$

enthalten alle $(x+1)$ als Primfaktor

6-Bit Character: $CRC - 12$
8-Bit Character: $CRC - 16, CRC - CCITT$

Prof. Bettina Schnor

445

446

CRC - Cyclic Redundancy Check

Definition 34:

Burstfehler der Länge j : \iff

$$\exists i > 0, a_i \in Z_2 : E(x) = x^i \cdot (x^{j-1} + a_{j-2} \cdot x^{j-2} + \dots + a_1 \cdot x^1 + 1)$$

Zur Prüfsummenberechnung kann die Polynomdivision mittels Schieberegister implementiert werden.

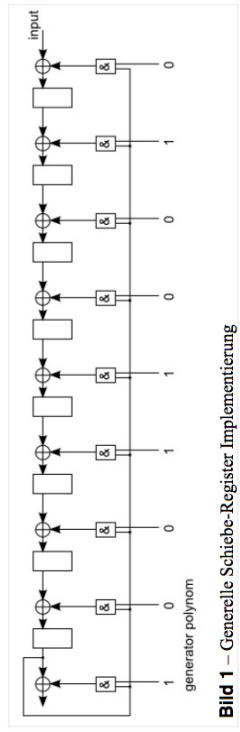


Bild 1 – Generelle Schiebe-Register Implementierung

- Bem.:
- 5) $CRC = 16$, $CRC - CCITT$ erkennen alle 1-Bit- und 2-Bit-Fehler,
alle Fehler mit ungerader Bitzahl
Burstfehler bis maximal Länge 16
99,997 % aller Burstfehler bis Länge 17
99,998 % aller Burstfehler bis Länge $j > 18$
 - 6) Berechnung der Checksum in Hardware mit Shift-Register
(Peterson, Brown 1961)

Prof. Bettina Schnor

CRC - Cyclic Redundancy Check

447

7.3 Flußsteuerung

Vereinfachung für ein fest vorgegebenes Generatorpolynom:

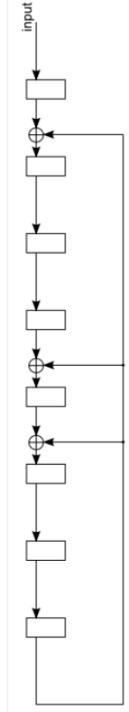


Bild 2 – Vereinfachte Schiebe-Register Implementierung

(Abbildung und Beispiel aus Werner Dichler, "CRC-Einführung", Version 0.0.1 vom 4.3.2013)

Probleme:

- 1 unzuverlässige Leitungen
- 2 Sender und Empfänger arbeiten i. d. R. nicht gleich schnell

Prof. Bettina Schnor

Prof. Bettina Schnor

449

Stop-and-Wait-Protokolle

Simplexprotokolle

Idee: dynamisches Anpassen der Übertragungsgeschwindigkeit mittels Quittungen (\simeq leerer Rahmen)

Rahmen geht verloren.
Quittung geht verloren.
nur Flussteuerung, keine Fehlerkontrolle

PAR-Protokolle (Positive Acknowledgement with Retransmission)

Simplexprotokolle

Idee: dynamisches Anpassen der Übertragungsgeschwindigkeit mittels Quittungen (\simeq leerer Rahmen)

Rahmen geht verloren.
Quittung geht verloren.
nur Flussteuerung, keine Fehlerkontrolle

Prof. Bettina Schnor

451

Optimierung bei Duplexprotokoll: Die Quittung wird nicht als eigener Rahmen verschickt, sondern im Header der Rückantwort notiert (sogenanntes Piggy Backing).

Prof. Bettina Schnor

452

Schiebefenster-Protokolle

- n -Bit Folgenummern: $0, 1, \dots, 2^n - 1$ werden modulo 2^n vergeben
- Das **Sendefenster (Empfangsfenster)** gibt die Nummern derjenigen Rahmen an, die aktuell verschickt (empfangen) werden dürfen.
- Der Empfänger schickt nach Empfang eines Rahmens eine Quittung.
Quittung $Q_k = k \iff$ Der Empfänger hat alle Rahmen bis einschließlich Nummer k erhalten.
- Für jeden abgeschickten Rahmen wird ein Timer gestartet.

Protokollschwächen:

- 1 Symmetrischer Datenverkehr wird vorausgesetzt.
- 2 ineffizient bei langen Laufzeiten (gilt generell für Stop-and-Wait-Protokolle):
Beispiel: Satellitverbindung mit 64 Kbps, Signalausbreitungsgeschwindigkeit: 270 ms Rahmen 1280 bit

Prof. Bettina Schnor

453 Prof. Bettina Schnor

454

7.4 High-level Data Link Control (HDLC)

1. Möglichkeit: Go-Back-N

- Der Empfänger nimmt Pakete nur in Reihenfolge an:
|Empfangsfenster| = 1
- Wird ein Rahmen mit falscher Rahmennummer empfangen, wiederholt der Empfänger die letzte Quittung.

2. Möglichkeit: Selective Repeat

- **Empfangsfenster > 1**
- Empfangsfenster muß klein genug sein, um Duplikate von neuen Rahmen unterscheiden zu können.
Im Fehlerfall: **selektives Wiederholen**

ISO-Protokoll

öffentliche Netze / WAN
stammt von IBM's SDLC (Synchronous Data Link Control) ab

verwendete Protokolle:
ANSI: Advanced Data Communication Control Procedure (ADCCP)
CCITT: Link Access Procedure-Balanced (LAP-B)

Grundlage für die Sicherungsschicht in

- Mobilfunktelefone über GSM
- Einwahl in das Internet mittels Point-to-Point-Protocol (PPP)
- CISCO Router: benutzen HDLC Serial Interfaces, HDLC over Layer 2 Tunneling Protocol (RFC 4349, 2006),
...
■ ...

Verbindungs-/Betriebsarten:

1) nichtbalancierte Konfiguration:

Primärstation := Leitstation, die den Datentransfer steuert, schickt **command-Steuerrahmen**.
Sekundärstation := Station arbeitet unter der Leitung der Primärstation und reagiert mit **response-Steuerrahmen**.

Mögliche Topologien:

- a) Punkt-zu-Punkt-Verbindung
- b) Mehrpunktverbindung

Betriebsart:

Normal Response Mode (NRM) := Die Primärstation initiiert den Datentransfer zur Sekundärstation. Die Sekundärstation kann Daten nur auf Anfrage der Primärstation hin senden. (Halbduplexbetrieb)
Asynchronous Response Mode (ARM) := Auch die Sekundärstation kann den Datentransfer initiieren, aber: Die Kontrolle bleibt bei der Primärstation. (Duplexbetrieb, Punkt-zu-Punkt-Verbindung)

Rahmenformat:	SD	Adresse 1 Byte	Control 1 Byte	Daten ≥ 0 Byte	Prüfsumme 2 Byte	ED
---------------	----	-------------------	-------------------	------------------------	---------------------	----

SD = ED = 0 11 11 11 0 mit Bitstopfen
Generatorpolynom für Prüfsumme: CRC-CITT

Das **Control-Feld** gibt u.a. den Rahmentyp an (Datenrahmen oder Steuerrahmen) und enthält Sende- und Quittungsfolgenummer.

⇒ Der HDLC-Header besteht aus 6 Byte.

Schiebefensterprotokoll: Sendefolgenummer (modulo 8), d.h. der Sender kann maximal 7 Datenrahmen schicken ohne eine Quittung erhalten zu haben. Quittungen werden default-mäßig piggyback verschickt.

N(R): = Sendefolgenummer des zuletzt erhaltenen Rahmens +1 („erwartete Rahmen“) quittiert alle Rahmen bis N(R)-1.

Bedeutung des Control-Felds

Rahmentyp	Befehle	Antwort	1	2	3	4	5	6	7	8
information	1	1	0	N(S)	P	N(R)				
supervisory		RR	1	0	0	0	P/F	N(R)		
supervisory		RNR	1	0	1	0	P/F	N(R)		
supervisory	REJ	1	0	0	1	P/F	N(R)			
	SREQ	1	0	1	1	P/F	N(R)			
unnumbered	SNRM	1	1	0	0	P	0	0	1	
unnumbered	SARM	1	1	1	1	P/F	0	0	0	
unnumbered	SABM	1	1	1	1	P	1	0	0	
unnumbered	DISC	1	1	0	0	P	0	1	0	
unnumbered		UA	1	1	0	0	F	1	1	0
unnumbered	CMDR	1	1	1	0	F	0	0	1	

Das fünfte Bit hat unterschiedliche Bedeutung: **Poll-/Final-Bit**.
Poll-Bit gesetzt ⇒ Sofort antworten, nicht „piggyback“.
Final-Bit gesetzt ⇒ kennzeichnet Quittungsrahmen.

Rahmentypen:

- **information frame:** Datenrahmen
- **unnumbered frame:** Steuerrahmen für Auf- und Abbau der Verbindung

SNRM	Aufrufbetrieb aufnehmen (set normal response mode)	RR	Station empfangsbereit (receive ready)
SARM	Konkurrenzbetrieb aufnehmen (set asynchronous response mode)	RNR	Station nicht empfangsbereit (receive not ready)
SABM	Konkurrenzbetrieb aufnehmen (set asynchronous balanced mode)	REJ	Aufforderung an Empfänger R, die Nachricht ab N(R) einschließlich nochmals an S zu übertragen (reject)
DISC	Abbruch der Verbindung (disconnect)	SREQ	Aufforderung an den Empfänger R, die Nachricht mit der Nummer N(R) nochmals an S zu übertragen (selective reject)
UA	nicht nummerierte Quittung für alle unnummerierten Steuerrahmen (unnumbered acknowledge)		
CMDR	negative Bestätigung der Kommandos (command reject)		

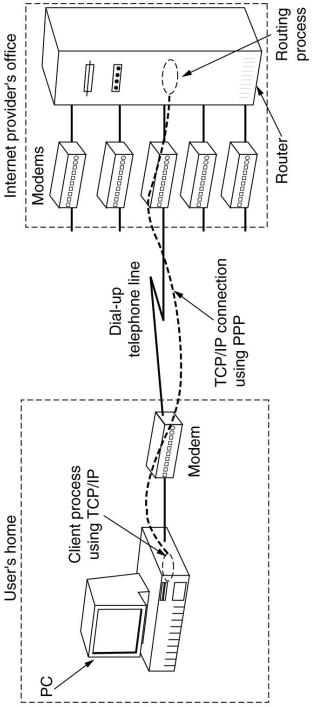
- **supervisory frame:** Steuerrahmen für Fluß- und Fehlerkontrolle

RR	Station empfangsbereit (receive ready)
RNR	Station nicht empfangsbereit (receive not ready)
REJ	Aufforderung an Empfänger R, die Nachricht ab N(R) einschließlich nochmals an S zu übertragen (reject)
SREQ	Aufforderung an den Empfänger R, die Nachricht mit der Nummer N(R) nochmals an S zu übertragen (selective reject)

7.5 PPP – Point-to-Point-Protocol

Das PPP Rahmenformat

De facto Standardprotokoll für die Einwahl in das Internet:



Einwahl über Modems RFC 1661, 1662, 1663: PPP über HDLC
heutzutage üblich: PPP over Ethernet (PPPoE)

Prof. Bettina Schnor

463

HDLC-
Rahmenformat:

Bytes	1	1	1	1 or 2	Variable	2 or 4	1
	Flag 01111110	Address 11111111	Control 00000011	Protocol 00000011	{Payload Checksum}	{Flag 01111110}	

- Es werden **keine** Adressen auf der Sicherungsschicht benutzt:
Address = 11111111 bedeutet eine Broadcast-Adresse.
- Control = 00000011 bedeutet, daß nur *unnumbered frames* geschickt werden, d.h. es wird **keine Flußkontrolle** durchgeführt.
(Erweiterungen für zuverlässigen Dienst in RFC 1663, aber selten benutzt).

464

Link Control Protocol (LCP)

Bytes	SD	Adresse	Control	Daten	Prüfsumme	ED
	1 Byte	1 Byte	1 Byte	≥ 0 Byte	2 Byte	

Aufgaben:

- Aktivieren und Testen von Leitungen,
- Verhandeln von Optionen, z.B.
 - Weglassen des Address- und Control-Felds,
 - Maximallänge der Payload (Default ist 1500 Bytes),
 - Größe des Protocol- und Checksum-Felds
 - Abbau der PPP-Verbindung
- PPP umfaßt zwei Teilprotokolle: **Link Control Protocol (LCP)** und **Network Control Protocol (NCP)**.
- Neu ist das **Protocol-Feld**, das angibt, welches Protokoll die Payload erzeugt hat. Es gibt definierte Codes für LCP, NCP, IP, IPv6, ...

Prof. Bettina Schnor

465

466

Prof. Bettina Schnor

Die LCP-Rahmentypen

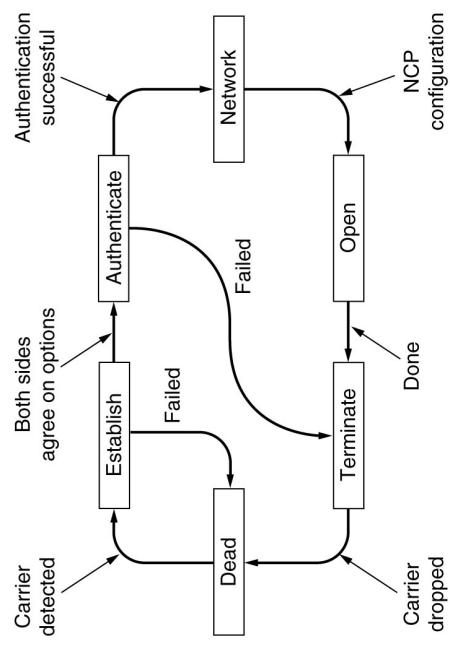
I steht für **Initiator**. R steht für **Responder**.

Name	Direction	Description
Configure-request	I → R	List of proposed options and values
Configure-ack	I ← R	All options are accepted
Configure-nak	I ← R	Some options are not accepted
Configure-reject	I ← R	Some options are not negotiable
Terminate-request	I → R	Request to shut the line down
Terminate-ack	I ← R	OK, line shut down
Code-reject	I ← R	Unknown request received
Protocol-reject	I ← R	Unknown protocol requested
Echo-request	I → R	Please send this frame back
Echo-reply	I ← R	Here is the frame back
Discard-request	I → R	Just discard this frame (for testing)

Network Control Protocol (NCP)

Für jedes unterstützte Protokoll der Vermittlungsschicht wird eine andere NCP-Version benötigt.
Das zu IP gehörige NCP-Protokoll ist z.B. für das dynamische Aushandeln von IP-Adressen zuständig.

Vereinfachte Darstellung der Phasen beim Auf- und Abbau von PPP-Verbindungen



6 Das ISO-Referenzmodell OSI

(International Standard Organization / Open Systems Interconnection)

Strukturierung der Kommunikationssoftware in **Schichten**:

Ein Prozeß auf Schicht n auf Rechner A kommuniziert mit einem Partnerprozeß auf Schicht n auf Rechner B. Die Regeln, nach denen die Kommunikation abläuft, werden das **Protokoll** von Schicht n genannt.

Die Kommunikation der Partnerprozesse aus Schicht n wird durch den Dienst der Schicht $n - 1$ realisiert. Der Dienst beinhaltet eine Menge von Funktionen, die Prozessen aus Schicht n von der Schicht $n - 1$ angeboten werden.

Die Schichten und Protokolle bilden die **Netzarchitektur**.

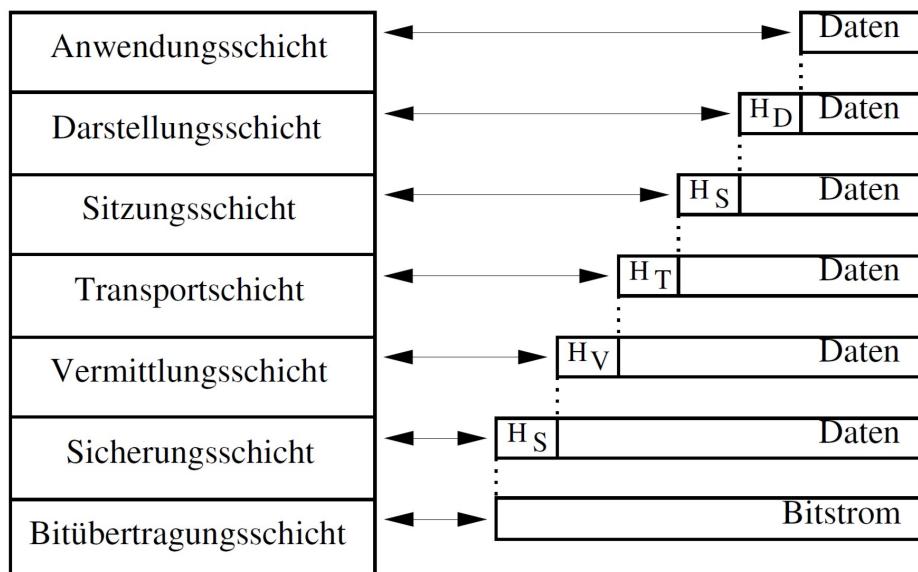


Abbildung 13: Aufbau des OSI Modells

6.1 Bitübertragungsschicht (Physical Layer)

Verschicken von Bitströmen \Rightarrow Normen für mechanische und elektrische Schnittstelle

6.2 Sicherungsschicht (Link Layer)

Die Dienstleistung der Bitübertragungsschicht ist die Übertragung einzelner Bits über ein (unzuverlässiges) Medium. Die Sicherungsschicht ist für die fehlerfreie Übertragung von Bitketten zwischen benachbarten Rechnern verantwortlich.

6.2.1 Aufgaben

1. Aufteilen der Daten in sogenannte Datenübertragungsrahmen (Frame) durch Einfügen von Rahmengrenzen.

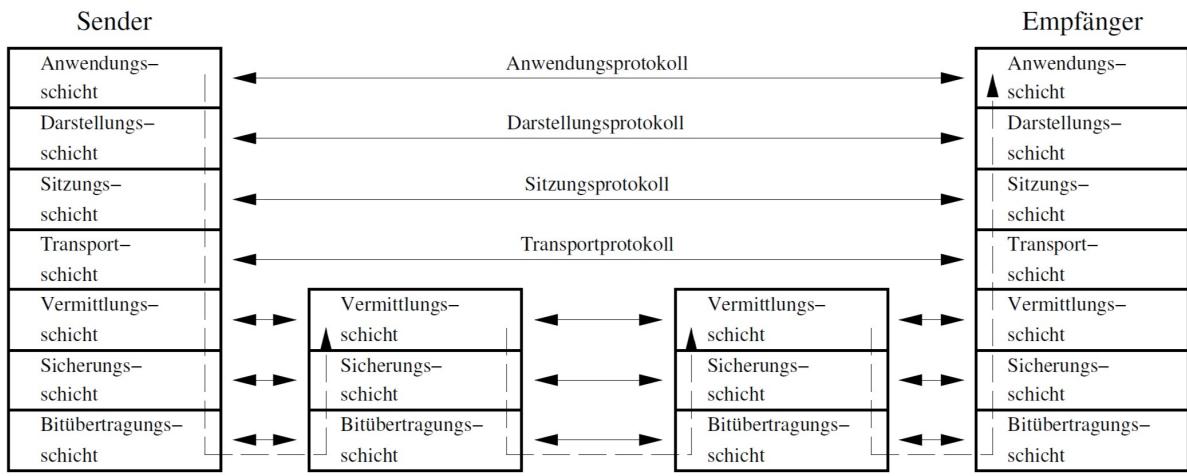


Abbildung 14: Kommunikationsfluss

2. Fehlererkennung und -korrektur
3. Flußkontrolle: Werden Nachrichten auf einem Link zwischen zwei Rechnern oder Vermittlungsknoten (Routern) verschickt, so kann es bei unterschiedlich leistungsfähigen Systemen zu einer Überlastung kommen und ggf. werden Pakete verworfen \Rightarrow Mittels Flußkontrolle wird die Senderate an die Verarbeitungsrate angepaßt.

6.3 Vermittlungsschicht (Network Layer)

Die Vermittlungsschicht ist verantwortlich für den Transport von Nachrichten von einem Quellrechner zu einem Zielrechner über beliebig, untereinander verbundene Datenleitungen und Teilnetze. Die Nachrichten müssen an den Zwischenstationen vermittelt werden.

6.3.1 Aufgaben

- Wegwahl (Routing)
- Anpassung bei Übergang an ein anderes Netzwerk (Adressierung, Paketgröße, ...)
- Flußkontrolle
- Abrechnung der Dienstleistung (Accounting)

6.3.2 Dienstarten

1. **Virtual Circuit Service (verbindungsorientierter Dienst):** Es wird zwischen Quell- und Zielrechner eine „perfekte“ logische (virtuelle) Verbindung aufgebaut und am Ende der Kommunikation wieder abgebaut.
Beispiel: X.25 PLP (Packet Layer Protocol)
2. **Datagram Service (verbindungsloser Dienst):** Die einzelnen Datenpakete einer Nachricht werden unabhängig voneinander vom Quell- zum Zielrechner übertragen. Jedes Datenpaket enthält Quell- und Zieladresse.
Beispiel: Internetprotokoll (IP)

6.4 Transportschicht (Transport Layer)

Das Transportprotokoll ist ein Ende-zu-Ende-Protokoll zwischen Quell- und Zielrechner.

Wesentliche Aufgabe der Transportschicht ist die Erhöhung der Dienstqualität des Vermittlungsdienstes:

- ggf. Paketverlust überwachen und korrigieren,
- ggf. Auf- und Abbau von Transportverbindungen
- Flußkontrolle zwischen Quell- und Zielrechner
- ggf. Aufteilen der Nachricht in kleinere Dateneinheiten für Schicht 3
- Multiplexing oder Splitting auf Verbindungen der Schicht 3

Beispiele: TCP (Transmission Control Protocol), UDP (User Datagram Protocol)

6.5 Sitzungsschicht (Session Layer)

Bereitstellung von Diensten, die von Anwendungen häufig benötigt werden. → Unterstützung von sogenannten Sitzungen

6.5.1 Beispiele für OSI-Dienste

- Checkpoints für zuverlässige Datenübertragung,
Beispiel: openFT von Fujitsu (Einsatzgebiet: Behörden- und Bankensektor),
- Token-Management für Videokonferenzen
- Client-Server-Kommunikation: Remote Procedure Call
RPC-Protokolle für synchrone Kommunikation zwischen Client und Server

6.6 Darstellungsschicht (Presentation Layer)

Die Schichten 1-5 befassen sich im wesentlichen mit der korrekten Übertragung von Bits vom Quell- zum Zielrechner.

Aufgabe der Schicht 6 ist die Bewahrung der Bits während der Übertragung.

- Anpassung an verschiedene Datenformate: Konvertierung zwischen verschiedenen Formaten
- Datenkomprimierung
- Verschlüsselung

6.7 Anwendungsschicht (Application Layer)

In der Anwendungsschicht laufen die Benutzerprogramme (Anwendungen).

Beispiele: Electronic Mail, Dateitransfer (ftp, http get/post), Online-Banking, eCommerce, ...

rlogin/rsh	NNTP	SMTP	Telnet	FTP	HTTP	NFS/ DNS,	SNMP	NTP
TCP						UDP		
IP								
Schicht 1 + 2								

- NNTP (Network News Transfer Protocol)
- SMTP (Simple Mail Transfer Protocol)
- DNS (Domain Name Service)
- SNMP (Simple Network Management Protocol)
- NTP (Network Time Protocol)
- NFS (Network File System)

Abbildung 15: Aufbau des Internet Protokoll Stacks

7 Die Sicherungsschicht

7.1 Aufgaben

7.1.1 Zusammenstellen der Rahmen

⇒ Kennzeichnen der Rahmengrenzen

- (a) mittels Anfangs- und Endflags (Begrenzer):

Rahmenformat: [SD | Steuerinf. von Schicht 2 | Daten | ED]

SD := Startdelimiter, ED := Enddelimiter

Beispiel: SD = ED = 0 11 11 11 0

Diese Zeichenkette darf in den Daten nicht vorkommen.

⇒ Bitstopfen := nach 5 aufeinander folgenden Einsen fügt der Sender eine 0 ein. Empfänger streicht nach 5 aufeinander folgenden Einsen eine Null heraus.

Beispiel:

Daten: 011011111011111001

Übertragen wird:

SD 011011111010111110001 ED

Beispiel: HDLC

- (b) Mischverfahren: SD + Feld mit Längenangabe ⇒ Schicht 2 muß Anzahl Bits zählen
 Beispiel: Ethernet

- (c) Verstöße gegen Kodierungsregeln

Beispiel: Tokenbus

7.1.2 Fehlerkontrolle

Link-to-Link-Überprüfung:

Wurde der Rahmen korrekt übertragen?

Ansatz: Sender berechnet und schickt Prüfsumme (Redundanzbits).

Empfänger berechnet ebenfalls Prüfsumme für den empfangenen Rahmen. „Stimmt“ die Prüfsumme, dann ist die Übertragung („höchstwahrscheinlich“) korrekt und die Daten werden an Schicht 3 weitergegeben. Andernfalls wird der Rahmen weggeworfen.

Beispiel: Hamming-Code, Cyclic Redundancy Code (CRC)

7.1.3 Flusssteuerung

Problem: Unterschiedlich schnell arbeitender Sender und Empfänger.

Falls der Empfänger langsamer arbeitet als der Sender, können beim Empfänger Datenpuffer überlaufen und Rahmen verloren gehen.

⇒ Protokolle zur Synchronisation des Datenflusses zwischen Sender und Empfänger, z.B. Stop-and-wait-Protokolle, Schiebefenster-Protokolle

7.2 Fehlererkennung und Fehlerkorrektur

Ansatz: Codewort der Länge $m + r$ (m Datenbits, r Redundanzbits)

7.2.1 Fehlerkorrekturcodes

2^m gültige Wörter

$$b_1 \dots b_m \xrightarrow{\text{codierung}} b_1 \dots b_m b_{m+1} \dots b_{m+r}$$

Fehler: $b_1 \dots \overline{b_i} \dots b_m$

Definition 28. Der Hamming-Abstand $h(b_1 \dots b_m, b'_1 \dots b'_m)$ ist die Anzahl der unterschiedlichen Bits, d.h. die Anzahl i mit $b_i \neq b'_i$.

Definition 29. Ein Übertragungsfehler heißt l-Bit-Fehler, falls $h(b_1 \dots b_m, r(b_1 \dots b_m)) = l$ gilt, wobei $r(b_1 \dots b_m)$ das empfangene Wort bezeichne.

Theorem 4. Gegeben sei eine Codierung C $C(b_1 \dots b_m) = b_1 \dots b_m b_{m+1} \dots b_{m+r}$

- Ist der minimale Hammingabstand aller gültigen Codewörter $d + 1$ so lassen sich alle l-Bit-Fehler, $l = 1 \dots d$ erkennen.
- Der minimale Hammingabstand aller Codewörter ist $2d + 1 \Rightarrow$ Es lassen sich alle l-Bit-Fehler, $l = 1 \dots d$ erkennen und korrigieren.

7.2.2 Beispiel für 1-Bit-Korrekturcode (Hamming 1950)

Das Datenwort wird aufgefüllt mit Redundanzbits (Checkbits) an den Positionen $b_i, i = 2^j, j = 0, 1, 2 \dots$

Checkbits sind Parity Bits:

Bit an Position k wird von allen Checkbits b_{2^j} geprüft, für die gilt, daß 2^j in der Dualzahldarstellung von k vorkommt.

Beim Empfangen werden alle Checkbits überprüft.

Falls Checkbit b_i falsch, setze counter = counter + i

Bit b_k verfälscht \rightarrow alle Checkbits b_i , die b_k mitprüfen sind verfälscht

counter = 0 \iff Codewort ist korrekt empfangen

counter = k \iff Bit b_k ist verfälscht

4-Bit Code generierung:	
Checkbits: p_i	$p_0 p_1 d_1 p_2 d_2 d_3 d_4$
Datenbits: d_j	$p_0 = d_1 \text{ xor } d_2 \text{ xor } d_4$
$p_0 p_1 d_1 p_2 d_2 d_3 d_4 p_3 d_5 d_6 d_7 d_8 d_9 d_{10} d_{11} p_4$ 1 2 3 4 ...	$p_1 = d_1 \text{ xor } d_3 \text{ xor } d_4$
	$p_2 = d_2 \text{ xor } d_3 \text{ xor } d_4$
<i>Bsp.: $d_3(\text{pos 6}) = 2 + 4 = 2^1 + 2^2$</i>	<i>Datenwort: 1011 -> 0110011</i>

Abbildung 16: Beispiel für Hamming-Korrektur

7.2.3 Fehlererkennung mittels Cyclic Redundancy Code (CRC)

zyklische Blocksicherung, Polynomcodes

$$b_1 \dots b_m \mapsto b_1 x^{m-1} + b_2 x^{m-2} + \dots + b_{m-1} x^1 + b_m$$

$$110001 \mapsto x^5 + x^4 + \dots + 1 =: M(x)$$

gegeben: „Generatorpolynom“ $G(x)$

Idee: Das zum Codewort zugehörige Polynom soll in $\mathbb{Z}_2[x]$ durch $G(x)$ teilbar sein.

Fehlererkennung: Das empfangene Wort wird auf das zugehörige Polynom abgebildet.

Geht Polynomdivision in $\mathbb{Z}_2[x]$ mit $G(x)$ auf?

Ja \rightarrow höchstwahrscheinlich kein Fehler

Nein \rightarrow Fehler!

7.2.4 Algorithmus zur Berechnung des Codeworts

1. $r := \text{Grad } (G(x))$
2. Polynomdivision mit Rest:
Bestimme $R(x)$ mit

$x^r \times M(x) = h(x) \times G(x) + R(x)$ in $\mathbb{Z}_2[x]$
 und $\text{Grad}(R(x)) \leq r$

3. Bestimme die zum Polynom

$$T(x) = x^r \times M(x) - R(x) = x^r \times M(x) + R(x)$$

gehörige Bitkette $b_1 \dots b_m \quad b_{m+1} \dots b_{m+r}$

Checksum \equiv Rest

7.2.5 Bemerkungen

1. Übertragungsfehler:

Empfangen wird $T(x) + E(x)$

$$(T(x) + E(x))/G(x) = h(x) + \frac{E(x)}{G(x)}$$

$G(x)|E(x) \Leftrightarrow$ Fehler wird nicht entdeckt

2. $(x + 1)|G(x) \Rightarrow$ Beliebige ungerade Anzahl von Bit-Fehlern wird erkannt.

3. Beispiele für Generatorpolynome:

$$\text{CRC - 12} = x^{12} + x^{11} + x^3 + x^2 + x + 1$$

$$\text{CRC - 16} = x^{16} + x^{15} + x^2 + 1$$

$$\text{CRC - CCITT} = x^{16} + x^{12} + x^5 + 1$$

Definition 30. Burstfehler der Länge $j \iff$

$$\exists i > 0, a_i \in \mathbb{Z}_2 : E(x) = x^i \times (x^{j-1} + a_{j-2} \times x^{j-2} + \dots + a_1 \times x^1 + 1)$$

7.3 Flusssteuerung

7.3.1 Kommunikationsarten zwischen Prozessen A, B

- simplex := unidirektional $A \rightarrow B$
- halbduplex := abwechselnd $A \rightarrow B$ und $B \rightarrow A$
- (voll-)duplex := bidirektional $A \leftrightarrow B$

Probleme:

- unzuverlässige Leitungen
- Sender und Empfänger arbeiten i. d. R. nicht gleich schnell

7.3.2 Stop-and-Wait-Protokolle

Simplexprotokolle

Idee: dynamisches Anpassen der Übertragungsgeschwindigkeit mittels Quittungen (\simeq leerer Rahmen)

Rahmen geht verloren.

Quittung geht verloren.

nur Flusssteuerung, keine Fehlerkontrolle

7.3.3 PAR-Protokolle (Positive Acknowledgement with Retransmission)

Optimierung bei Duplexprotokoll: Die Quittung wird nicht als eigener Rahmen verschickt, sondern im Header der Rückantwort notiert (sogenanntes Piggy Backing).

7.3.4 Protokollschwächen

1. Symmetrischer Datenverkehr wird vorausgesetzt.
2. ineffizient bei langen Laufzeiten (gilt generell für Stop-and-Wait-Protokolle)
Beispiel: Satellitverbindung mit 64 Kbps,
Signalausbreitungsgeschwindigkeit: 270 ms
Rahmen 1280 bit

7.3.5 Schiebefenster-Protokolle

- n-Bit Folgenummern: $0, 1, \dots, 2^n - 1$ werden modulo 2^n vergeben
- Das Sendefenster (Empfangsfenster) gibt die Nummern derjenigen Rahmen an, die aktuell verschickt (empfangen) werden dürfen.
- Der Empfänger schickt nach Empfang eines Rahmens eine Quittung.
Quittung $Q_k = k \Leftrightarrow$ Der Empfänger hat alle Rahmen bis einschließlich Nummer k erhalten.
- Für jeden abgeschickten Rahmen wird ein Timer gestartet.

7.3.6 1. Möglichkeit: Go-Back-N

Der Empfänger nimmt Pakete nur in Reihenfolge an: $|\text{Empfangsfenster}| = 1$

Wird ein Rahmen mit falscher Rahmennummer empfangen, wiederholt der Empfänger die letzte Quittung.

7.3.7 2. Möglichkeit: Selective Repeat

Empfangsfenster > 1

Empfangsfenster muß klein genug sein, um Duplikate von neuen Rahmen unterscheiden zu können.
Im Fehlerfall: selektives Wiederholen

7.4 High-level Data Link Control (HDLC)

7.4.1 Verbindungs-/Betriebsarten

7.4.2 nichtbalancierte Konfiguration

Primärstation := Leitstation, die den Datentransfer steuert, schickt command-Steuerrahmen.
Sekundärstation := Station arbeitet unter der Leitung der Primärstation und reagiert mit response-Steuerrahmen.

Mögliche Topologien:

- a) Punkt-zu-Punkt-Verbindung
- b) Mehrpunktverbindung

Betriebsart:

Normal Response Mode (NRM) := Die Primärstation initiiert den Datentransfer zur Sekundärstation. Die Sekundärstation kann Daten nur auf Anfrage der Primärstation hin senden. (Halbduplexbetrieb)

Asynchronous Response Mode (ARM) := Auch die Sekundärstation kann den Datentransfer initiieren, aber: Die Kontrolle bleibt bei der Primärstation. (Duplexbetrieb, Punkt-zu-Punkt-Verbindung)

7.4.3 balancierte Konfiguration

Kombinierte Stationen, die sowohl die Funktion einer Primärstation als auch einer Sekundärstation ausüben.

Betriebsart:

Asynchronous Balanced Mode (ABM) := Jede der beiden Stationen kann den Datentransfer initiieren. (Duplexbetrieb)

7.4.4 Rahmenformat

[SD | Adresse (1 B) | Control (1 B) | Daten (≥ 0 B) | Prüfsumme (2 B) | ED]

SD = ED = 0 11 11 11 0 mit Bitstopfen

Generatorpolynom für Prüfsumme: CRC-CCITT

Das Control-Feld gibt u.a. den Rahmentyp an (Datenrahmen oder Steuerrahmen) und enthält Sende- und Quittungsfolgenummer.

⇒ Der HDLC-Header besteht aus 6 Byte.

7.4.5 Rahmentypen

- information frame: Datenrahmen
- unnumbered frame: Steuerrahmen für Auf- und Abbau der Verbindung
- supervisory frame: Steuerrahmen für Fluss- und Fehlerkontrolle

7.4.6 Schiebefensterprotokoll

Sendefolgennummer (modulo 8), d.h. der Sender kann maximal 7 Datenrahmen schicken ohne eine Quittung erhalten zu haben. Quittungen werden default-mäßig piggyback verschickt.

N(R): = Sendefolgennummer des zuletzt erhaltenen Rahmens +1 („erwartete Rahmen“) quittiert alle Rahmen bis N(R)-1.

7.5 PPP – Point-to-Point-Protocol

De facto Standardprotokoll für die Einwahl in das Internet

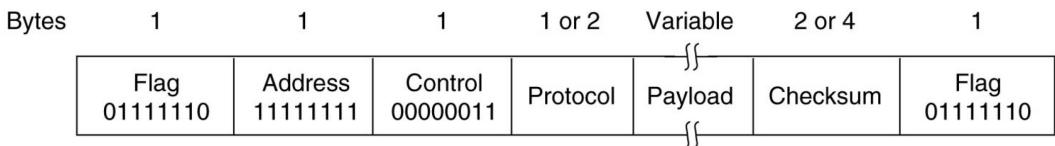


Abbildung 17: PPP Rahmenformat

7.5.1 PPP Rahmenformat

- Es werden keine Adressen auf der Sicherungsschicht benutzt:
Address = 11111111 bedeutet eine Broadcast-Adresse.
- Control = 00000011 bedeutet, daß nur unnumbered frames geschickt werden, d.h. es wird keine Flußkontrolle durchgeführt.
- PPP umfaßt zwei Teilprotokolle:
Link Control Protocol (LCP)
und Network Control Protocol (NCP).
- Neu ist das Protocol-Feld, das angibt, welches Protokoll die Payload erzeugt hat. Es gibt definierte Codes für LCP, NCP, IP, IPv6, ...

7.5.2 Link Control Protocol (LCP)

7.5.3 Aufgaben

- Aktivieren und Testen von Leitungen,
- Verhandeln von Optionen, z.B.
Weglassen des Address- und Control-Felds,
Maximallänge der Payload (Default ist 1500 Bytes),
Größe des Protocol- und Checksum-Felds
Abbau der PPP-Verbindung

7.5.4 Die LCP-Rahmentypen

7.5.5 Network Control Protocol (NCP)

Für jedes unterstützte Protokoll der Vermittlungsschicht wird eine andere NCP-Version benötigt. Das zu IP gehörige NCP-Protokoll ist z.B. für das dynamische Aushandeln von IP-Adressen zuständig.

8 Der IEEE-Standard 802 für lokale Netze

8.1 Ethernet

8.1.1 Ethernet Paketformat (1983-1996)

Mindestlänge von Zieladresse bis zur Prüfsumme: 64 Bytes ⇒ Padding
Prüfsumme: CRC-32

Name	Direction	Description
Configure-request	I → R	List of proposed options and values
Configure-ack	I ← R	All options are accepted
Configure-nak	I ← R	Some options are not accepted
Configure-reject	I ← R	Some options are not negotiable
Terminate-request	I → R	Request to shut the line down
Terminate-ack	I ← R	OK, line shut down
Code-reject	I ← R	Unknown request received
Protocol-reject	I ← R	Unknown protocol requested
Echo-request	I → R	Please send this frame back
Echo-reply	I ← R	Here is the frame back
Discard-request	I → R	Just discard this frame (for testing)

Abbildung 18: LCP Rahmentypen - I steht für Initiator. R steht für Responder.

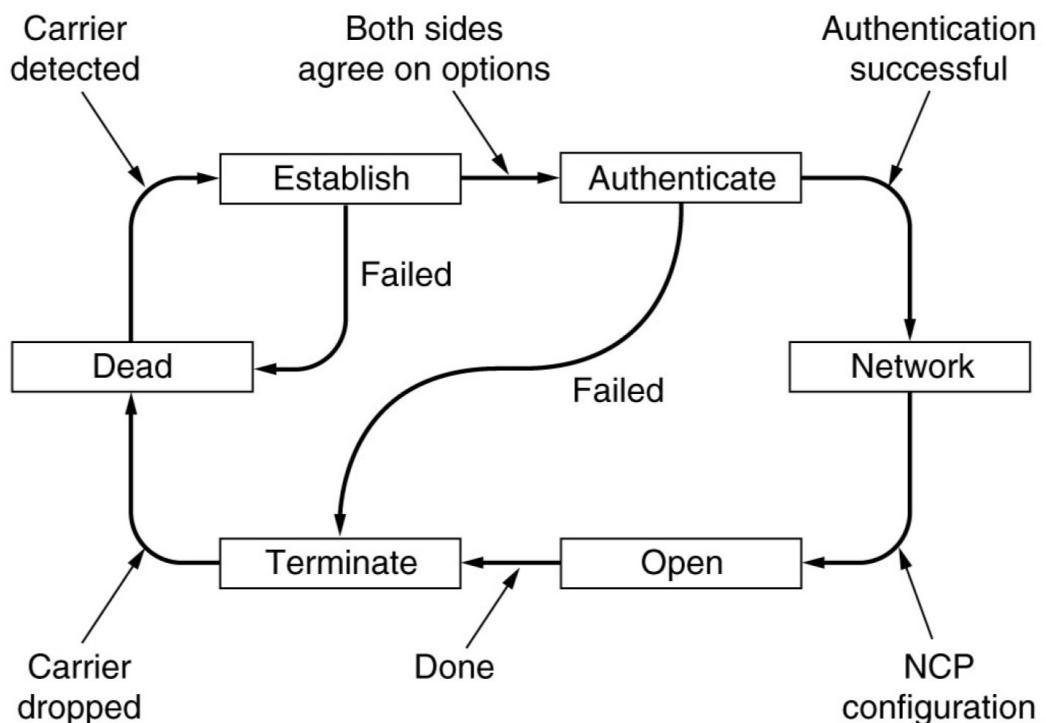


Abbildung 19: Vereinfachte Darstellung der Phasen beim Auf- und Abbau von PPP-Verbindungen

7	1	6	6	2	0-1500	0-46	4
Präambel	Start-begr.	Ziel-adr.	Quell-adr.	Länge/Typ	Daten	Pad	Prüfsumme

Abbildung 20: Ethernet Paketformat

8.1.2 CSMA/CD-Verfahren (Carrier Sense Multiple Access with Collision Detection)

- Die sendewillige Station hört das Kabel ab und sendet, falls das Kabel frei ist (Carrier Sense).
- Andernfalls wartet die Station, bis das Kabel frei wird (1-persistent).
- Da alle sendewilligen Stationen gleichberechtigt sind (Multiple Access), kann es zu Kollisionen kommen. Jeder Sender hört deshalb auch die eigene Nachricht mit und überprüft, ob sie ungestört bleibt (Collision Detection).
- Im Fehlerfall wird ein Jamming-Signal geschickt und der Sendevorgang nach einem zufällig gewählten Zeitintervall wiederholt (Binary Exponential Backoff Algorithm).

⇒ Random-Zugriffsverfahren

8.1.3 Binary Exponential Backoff Algorithm

- Zeit wird in “Slots” eingeteilt: slot = $2 t_p$
- nach 1.Kollision: 0 oder 1 Slot warten
- nach 2.Kollision: 0, 1, 2 oder 3 Slots warten
- allg.: nach i-ter Kollision: 0, 1, ... oder $2^i - 1$ Slots warten
- ab 10.Kollision: 0, 1, oder 1023 Slots warten
- Abbruch nach 16.Kollision und Fehlermeldung an nächsthöhere Schicht

8.1.4 Eigenschaften CSMA/CD

- Random-Zugriffsverfahren, daher nicht für Echtzeitanforderungen geeignet
- Ethernet spezifiziert eine Bitfehlerrate (Bit Error Rate (BER)) von 10^{-8} im Worst-Case für Kupfer. Bei Sprachübertragung ist eine BER von 10^{-6} und kleiner hinreichend.
- Gigabit Ethernet spezifiziert eine BER von 10^{-10} , und 10 Gigabit Ethernet von 10^{-12} .
- Pakete können wegen Überlast im Switch oder von der Karte weggeworfen werden. Dies wird vom Protokoll nicht bemerkt und folgerichtig auch nicht behoben. ⇒ keine Flusskontrolle. Flusskontrolle muss in höheren Schichten erfolgen.
- FIFO-Reihenfolge der Pakete zwischen zwei Kommunikationspartnern wird eingehalten.

8.2 Digital Subscriber Line (DSL)

In DSL-Netzen sammeln sogenannte DSL-Access Multiplexer (DSLAMs) den Datenverkehr der vor Ort angeschlossenen Teilnehmer und leiten ihn an einen Router, den sogenannten DSL Access Concentrator (DSL-AC) weiter. Der DSL-AC ist also der erste Hop auf dem Weg in das Internet.

8.3 PPP over Ethernet (PPPoE)

PPPoE besteht aus einer Discovery- und einer Session-Phase. Während der Discovery-Phase wird die Ethernet-Adresse des Servers gelernt und eine eindeutige Session-ID ausgehandelt: “In the Discovery process, a Host (the client) discovers an Access Concentrator (the server). Based on the network topology, there may be more than one Access Concentrator that the Host can communicate with. The Discovery stage allows the Host to discover all Access Concentrators and then select one.” Interpretation des Länge/Typ-Feldes:

7	1	6	6	2	6	4	
Präambel	Startbegr.	Ziel-adr.	Quell-adr.	Type = 0x8863 bzw. 0x8864	PPPoE-Paket-Header	Daten oder Padding	Prüfsumme

Abbildung 21: PPPoE Paketformat

The ETHER_TYPE is set to either 0x8863 (Discovery Stage) or 0x8864 (PPP Session Stage).

8.3.1 Discovery-Stage

1. Der Host/Client schickt ein Initiation Packet an die Broadcast-Adresse.
2. Einer oder mehrere Access Concentrators antworten mit Offer Packet.
3. Der Host/Client wählt einen Access Concentrator aus. “The choice can be based on the AC-Name or the Services offered.”
4. Der Host/Client schickt ein Unicast Session Request Packet an den ausgewählten Access Concentrator mit SESSION_ID=0x0000.
5. Der Access Concentrator erzeugt für diese Session eine eindeutige Session-ID und schickt diese mit einem Confirmation Packet an den Host/Client.

8.3.2 Session-Stage

Sind Ethernet-Adresse und Session-ID ausgehandelt, beginnt die Session-Phase. Ab jetzt werden PPP-Pakete in der Ethernet-Payload verschickt (PPP Encapsulation).

8.4 VLAN-Technologie mittels Ethernet

Das lokale Netz wird in virtuelle Netze aufgeteilt. Ziel ist es Netzwerksegmente zu separieren. IEEE 802.1Q unterstützt virtuelle LANs (VLANs) in einem 802.3 Ethernet. Switches müssen diese Funktionalität ebenfalls unterstützen.

8.4.1 802.1Q Tag Format

- Tag protocol identifier (TPID) (16 Bit): An der Position, an der sonst das EtherType/Länge-Feld steht. Wird auf TPID = 0x8100 gesetzt, was bedeutet, dass es sich um einen Ethernetrahmen gemäß 802.1Q handelt. Das eigentliche EtherType/Länge-Feld wird um 4 Bytes verschoben und die Länge der Payload um 4 Byte reduziert.

- Byte 3 und 4 sind Tag control information (TCI) und werden in PCP, DEI und VID unterteilt.
- Priority code point (PCP) (3 Bit): Gibt den Frame Priority Level an gemäß IEEE 802.1p.
- Drop eligible indicator (DEI) (1 Bit): Kennzeichnet Ethernetrahmen, die im Falle von Überlastsituationen gedroptt werden dürfen.
- VLAN identifier (VID) (12 Bit)

9 Die Vermittlungsschicht

9.1 Wegwahlverfahren (Routing) und Addressierung

Minimalinformation eines Knotens: Adresse der Nachbarknoten

Fluten: Das Paket wird auf allen Leitungen mit Ausnahme der Empfangsleitung gesendet.

- + einfaches Verfahren
- + Paket findet den Weg zum Empfänger

9.1.1 Ziele eines Wegwahlverfahrens

1. Übertragungszeit minimieren, d.h. den „schnellsten“ Weg vom Sender zum Empfänger bestimmen,
Der schnellste Weg hängt ab von:
 - (a) Anzahl Vermittlungsrechner auf dem Weg (Anzahl Hops)
 - (b) Leitungslänge und Bandbreite,
 - (c) Auslastung der Vermittlungsrechner.
2. Gesamtdurchsatz maximieren,
3. Übertragungskosten minimieren.

9.1.2 Klassifikation von Wegwahlverfahren

1. **statische Wegwahlverfahren:** Wege werden im voraus berechnet und in sogenannten Wegwahltabellen gespeichert.
Mögliche Parameter:
 - (a) Netztopologie
 - (b) Leitungslänge und Bandbreite
 - (c) erwartete Auslastung der Leitungen
2. **adaptive Wegwahlverfahren:** Wegwahl passt sich der aktuellen Netzsituation an:
 - (a) Störungen (Leitungs-/Vermittlungsrechnerausfall)
 - (b) aktuelle Lastsituation

9.1.3 Link-State-Routing-Algorithmus

Ein Link-State-Routing-Algorithmus setzt die komplette Kenntnis des Netzwerk-Graphen voraus.
⇒ Jeder Router kann die kürzesten Wege berechnen und damit seine Wegwahltabelle bestimmen.
Bestimmung des kürzesten Weges ⇒ Dijkstra-Algorithmus

Da volle Kenntnis des Netzwerk-Graphen vorausgesetzt wird, ist dieses Routingverfahren meist nur innerhalb des Netzes eines Netzwerkbetreibers anwendbar.

9.1.4 [DIJKSTRA-ALGORITHMUS!]

9.1.5 Hierarchische Addressierung

Eine hierarchische Adressierung ermöglicht kleinere/effizientere Wegwahltabellen.

Beispiel:

Internet-Adressen sind 32-Bit Adressen, die aus zwei Teilen bestehen: net-id | host-id

übliche Schreibweise: gepunktete Dezimalnotation, z.B. 141.83.21.121

Rechner mit mehreren Netzverbindungen (z.B. Gateways) haben mehrere IP-Adressen.

9.1.6 Beispiele für adaptive Verfahren

Isolierte (lokale) Verfahren von Baran (1964): Jeder Vermittlungsrechner entscheidet selber aufgrund von selbstgesammelten Informationen.

9.1.7 1. Hot-Potato-Routing

Die empfangene Nachricht wird so schnell wie möglich weitergegeben, d.h. an den Nachbarn mit der kürzesten Warteschlange vor der Ausgangsleitung.

Modifikation: Wegwahltabelle, die nicht nur die kürzesten Wege, sondern auch Alternativwege enthält. Unter diesen wird dann gemäß Warteschlangenlänge entschieden.

9.1.8 2. Backward Learning

Die Pakete zählen unterwegs die Anzahl Vermittlungsrechner, die sie passieren (hopscount). Empfängt ein Rechner A über den Nachbarknoten N ein Paket von Rechner S mit hopscount = k, lernt A, daß es einen Weg der Länge k zu S über N gibt und ergänzt seine Wegwahltabelle

Regeln:

1. (Wegwahl) Ist der Empfänger in der Wegwahltabelle eingetragen, wird das Paket an den zuständigen Nachbarknoten geschickt.
Sonst: Fluten
2. (Update) Die Wegwahltabelle wird aktualisiert, falls ein Paket eintrifft, das einen kürzeren Weg angibt.

9.1.9 3. Vector Distance Routing

Austausch von Übertragungszeitvektoren:

$\tilde{d}(A, Z_i)$ gibt die aktuelle Schätzung für die Übertragungszeit von A zum Ziel Z_i an (z.B. in Form

der Hop-Count-Metrik).

$\tilde{d}(A, Z_i) = \infty$, falls kein Weg bekannt ist.

1. Initialisierung der Wegwahltabellen z.B. mittels Dijkstra-Algorithmus oder Backward Learning
2. Knoten A misst periodisch die Laufzeit von Nachrichten zu seinen Nachbarknoten N: $d(A, N)$
3. A schickt entweder in regelmäßigen Zeitabständen (synchron) oder nach einer Aktualisierung (asynchron) den Übertragungszeitvektor an alle seine Nachbarn.
4. Neuberechnung der Wege: Hat A die Übertragungsvektoren aller seiner Nachbarn erhalten, so wird für jeden Zielknoten Z und jeden Nachbarn N die geschätzte Weglänge $d(A, Z)$ über N neu berechnet:
$$d_N(A, Z) := d(A, N) + \tilde{d}(N, Z) \text{ und}$$
$$d(A, Z) := \min\{d_N(A, Z) | N \text{ Nachbar von } A\} \text{ bestimmt.}$$

9.2 Fragmentierung

Definition 31. Paketfragmentierung: Das Datenpaket aus Netz 1 wird in mehrere Datenpakete zerlegt, um der Rahmengröße von Netz 2 zu genügen.

9.2.1 Transparente Fragmentierung

Jedes Fragment wird an das gleiche „Ausgangsgateway“ adressiert. Dort werden die Fragmente wieder zusammengesetzt, bevor sie ggf. in das nächste Netz geschickt werden.

- + Netze, in denen das Paket nicht fragmentiert werden muß, durchläuft das Paket als eine Nachricht. =) weniger fehleranfällig
 - Pro Netz muß ggf. einmal fragmentiert und reassembliert werden
- ⇒ bei virtuellen Verbindungen geeignet

9.2.2 Internetfragmentierung

für Datagrammnetzwerke: Fragmente werden erst am Zielrechner wieder zusammengesetzt.

- + minimaler Reassemblierungsaufwand
- + flexible Wegwahl: Fragmente können unterschiedliche Ausgangsgateways benutzen
- Header muss auf alle Fragmente kopiert werden
- Fehlerkontrolle liegt beim Zielrechner

9.3 Internet Protocol (IP)

9.3.1 Kennzeichen

- Schicht 3 Protokoll,
- Datagram Service,
- Wegwahl, Anpassen der Paketgröße, Adressen etc. beim Übergang in anderes Netz, Flusskontrolle,
- hierarchische Wegwahl: Routing orientiert sich am Zielnetzwerk, nicht am Zielrechnerx

9.3.2 IP Forward-Algorithm: RouteDatagram (Datagram, Wegwahltabelle)

1. Bestimme die Ziel-IP-Adresse des Datagramms D und den zugehörigen Netzwerk-Präfix N
2. IF N ist ein direkt verbundenes Netzwerk
 THEN schicke das Datagramm auf dem zugehörigen Interface raus
 ELSE IF Wegwahltabelle besitzt einen Eintrag für Ziel-IP
 THEN schicke Datagramm zum nächsten Router gemäß Eintrag
 ELSE IF Wegwahltabelle besitzt einen Eintrag für Netzwerk N
 THEN schicke das Datagramm zum nächsten Router gemäß Eintrag
 ELSE IF Wegwahltabelle enthält Default-Eintrag
 THEN schicke das Datagramm zum Default-Router gemäß Eintrag
 ELSE Routing-Error!

9.3.3 IP Packet Processing - Outgoing Packet

What happens when the Kernel sends an IP-Packet?

1. Fill in Destination IP-Address
The application supplies the Destination IP-Address (numeric Address or Hostname). This usually requires a DNS-Lookup, in order to map a Hostname to an IP-Address
2. Fill in Source IP-Address
If a host has only one configured interface/address, the IP-Address of this interface is used.
If a host has multiple addresses the kernel performs a Route-Lookup over the routing table to determine the closest matching route to the destination. The source of this route becomes the Source IP-Address.
3. Fill in Source HW-Address
The Hardware(MAC)-Address of the Source Interface is used.
4. Fill in Destination HW-Address
The Network Address of the IP-Destination is compared with the local IP-Address (Subnet-Mask) in order to determine if the Destination is local or remote.

9.3.4 IP Wegwahltabellen

Definition 32. Gateway Protocol

1. Eine Menge von Netzen und Routern, die administrativ zusammengehören, heißen Autonomes System. Mittels des Exterior Gateway Protocol (EGP) findet Wegwahl zwischen Autonomen Systemen statt.
2. Wege innerhalb eines autonomen Systems werden mit dem Interior Gateway Protokoll (IGP) bestimmt.

9.3.5 Internet Control Message Protocol (ICMP)

ICMP legt Regeln zur Kommunikation zwischen verschiedenen IP-Instanzen fest zum Austausch von Fehlernachrichten (z.B. destination unreachable, route change request) und Statusmeldungen (z.B. echo request)

ICMP ist Teil von IP. ICMP-Nachrichten werden als IP-Nachricht verschickt (Tunneling).

Wie erkennt IP, dass es sich um ein ICMP-Paket handelt?

protocol = 1 \Rightarrow ICMP-Nachricht

9.3.6 Beispiel

ping: Kommandozeilen-Tool, das eine Folge von ICMP-Nachrichten vom Typ Echo Request schickt und ICMP Echo Reply Nachrichten als Antwort bekommt.

Teil II

Hausaufgaben

Blatt 1

Aufgabe 1.1: Unix-Stammbaum

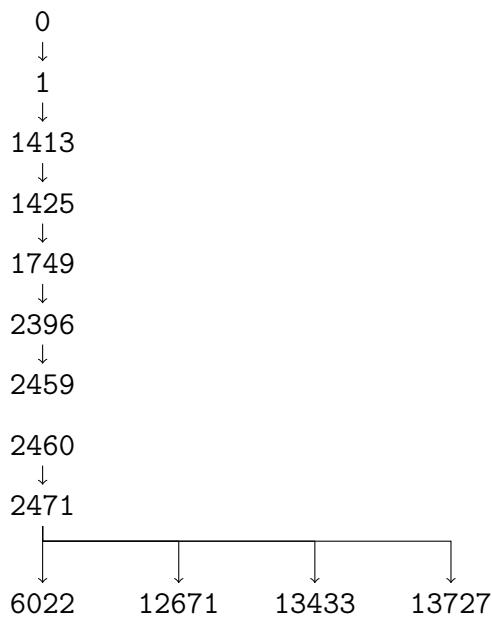
- a) UNICS → UNIX → Minix → Linux
- b) Das Diagramm lässt darauf schließen, dass Linux zwar nach Unix und Minix entstand, jedoch nicht aus ihnen hervorging.

Aufgabe 1.2: Betriebssysteme

1. Linux dominiert die Liste, da es durch seine zahlreichen Optimierungs- und Anpassungsmöglichkeiten ein sehr individuelles Betriebssystem ist. Außerdem läuft es deutlich stabiler als andere Betriebssysteme und ist nicht so fehleranfällig.
2. *Nova* wird seit 2005 von der Universidad de Las Ciencias Informaticas (UCI) in Havanna entwickelt. Es basiert auf Ubuntu, greift jedoch zu 80% auf Debian Pakete zurück. Wegen des Wirtschaftsembargo der USA gibt es in Kuba kein Windows über den legalen Weg. Außerdem konnte nach Aussage des Dekans der UCI die Sicherheit von Windows mangels Zugang zum Quellcode nicht überprüft werden. 2004 wurde deshalb der Umstieg der kubanischen IT Infrastruktur auf freie Software beschlossen. Im darauf folgenden Jahr begann die Entwicklung von *Nova*.

Aufgabe 1.3: Prozessverwaltung

- UID User ID des Prozessbesitzers/- erstellers
- PID Prozessnummer unter die der Prozess läuft
- PPID Elternprozessnummer, zeigt die PID des Prozesse an, von dem ein Prozess abstammt
- C CPU-Gebrauch und Scheduling Information
- STIME Zeit, wann der Prozess gestartet wurde
- TTY Nummer des Terminals oder der virtuellen Console zur Ausgabe
- TIME CPU-Zeit, die ein Prozess benutzt hat
- CMD Kommando, mit dem Prozess gestartet wurde



Abstammung des Prozesses 6022 mit Geschwisterprozessen

Aufgabe 1.4: Prozesstabellen

siehe Programmcode

Aufgabe 1.5: Prozesszustände

In einem System existieren zunächst keine rechenbereiten Prozesse. Es kommen drei rechenbereite Prozesse A, B und C hinzu. Nacheinander treten die folgenden Ereignisse ein:

1. Der Scheduler entscheidet, dass A als nächster Prozess die CPU erhält.
2. Prozess A will Daten über die Netzwerkkarte (I/O) empfangen.
3. Die Zeitscheibe des aktuell rechnenden Prozesses läuft ab.
4. Der I/O-Vorgang von Schritt 2 ist beendet und der I/O-Interrupt wurde ausgelöst.

Geben Sie für jeden Prozess und für jedes Ereignis, den letzten Prozesszustand vor und den ersten Prozesszustand *nach* der Aktivierung des Schedulers an!

Erklären Sie jeweils kurz, warum Sie sich für die gewählten Zustände entschieden haben!

Lösung:

Ereignisse	Prozesse		
	A	B	C
vor 1.	rechenbereit	rechenbereit	rechenbereit
nach 1.	rechnend	rechenbereit	rechenbereit
vor 2.	blockiert	rechenbereit	rechenbereit
nach 2.	blockiert	rechnend	rechenbereit
vor 3.	blockiert	rechenbereit	rechenbereit
nach 3.	blockiert	rechenbereit	rechnend
vor 4.	rechenbereit	rechenbereit	rechnend
nach 4.	rechnend	rechenbereit	rechenbereit

Vor 1. sind gemäß Aufgabenstellung alle drei Prozesse rechenbereit. Bei 1. erhält A die CPU. Bei 2. wird A blockiert bis I/O von Netzwerkkarte empfangen wird (dann bei 4.) und ein anderer Prozess kann in der Zwischenzeit die CPU erhalten (B oder C).

Bei 3. läuft die Zeitscheibe von B oder C ab, je nachdem welchen Prozess man vorher rechnen ließ. Der jeweils andere darf nun rechnen. Bei 4. kommt der I/O Interrupt und unterbricht den vorübergehend rechnenden Prozess und A bekommt wieder die CPU.

Aufgabe 1.6: Deadlocks

Das klassische Synchronisationsproblem der fünf speisenden Philosophen lautet wie folgt:

In einem Elfenbeinturm leben fünf Philosophen. Der Tagesablauf eines jeden Philosophen besteht abwechselnd aus Essen und Denken. Die fünf Philosophen essen an einem runden Tisch, auf dem in der Mitte eine immer gefüllte Schüssel mit Reis steht. Jeder Philosoph hat seinen festen Platz am Tisch und zwischen zwei Plätzen liegt genau ein Essstäbchen.

Das Problem der Philosophen besteht nun darin, dass der Reis nur mit zwei Stäbchen zu essen ist. Darüber hinaus darf jeder Philosoph nur das direkt rechts und das direkt links neben ihm liegende Stäbchen zum Essen benutzen. Das bedeutet, dass zwei benachbarte Philosophen nicht gleichzeitig essen können.

Erläutern Sie anhand dieses Philosophen-Beispiels die Begriffe *Deadlock*, *Livelock* und *Starvation!* Gehen Sie davon aus, dass die Philosophen parallel zueinander arbeiten!

Lösung:

Ein Philosoph kann verhungern wenn entweder mindestens einer seiner Tischnachbarn ständig isst oder sie sich beide abwechseln. Denn er bekäme nur ein oder gar kein Stäbchen.

Ein Deadlock wäre, wenn alle Philosophen gleichzeitig das rechte oder das linke Stäbchen nehmen würden. Jeder hätte nur ein Stäbchen und würde warten, dass der Andere seins ablegt.

Livelock wäre, wenn zwei benachbarte Philosophen gleichzeitig nach beiden Stäbchen zu greifen versuchen und das mittige Stäbchen keiner nimmt, weil sie zum arbeiten ausweichen. Und das immer wieder.

Aufgabe 1.7: Fork

Wie viele Prozesse werden bei Ausführung dieses Programms erzeugt?

```

int main(void) {
    int i;
    for (i = 0; i < 4; i++) {
        fork();
    }

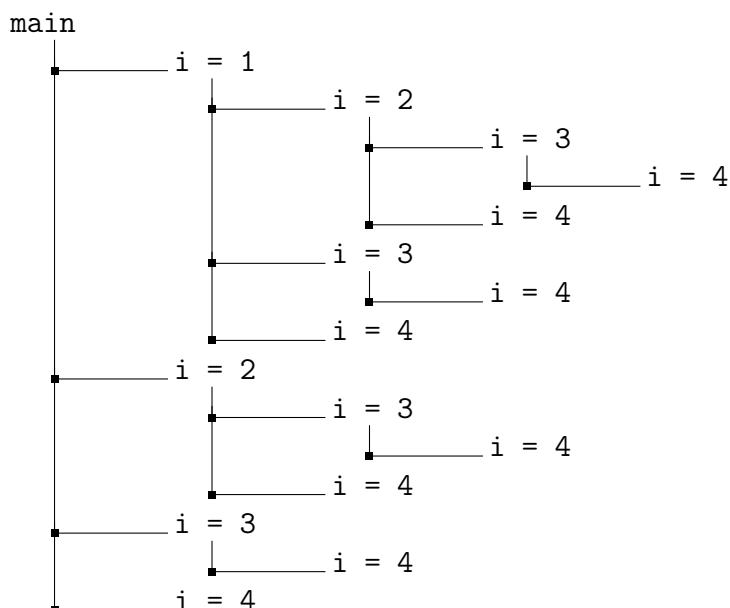
    return 0;
}

```

Erstellen Sie das zugehörige Prozessabstammungsdiagramm, das den ersten gestarteten Prozess und alle Kindprozesse enthält!

Lösung

Bei der Ausführung dieses Programmes werden $2^4 = 16$ Prozesse erzeugt:



Blatt 2

Aufgabe 2.1: Prozesserzeugung

Programmieraufgabe

Aufgabe 2.5: POSIX-Threads

Programmieraufgabe

Aufgabe 2.2: Echtzeit-Scheduling

Gegeben sei mit ihren Planungsdaten die folgende Menge an Rechenaufträgen eines Einkern-Echtzeitsystems:

Prozess	Ankunftszeit	Rechenzeit	Deadline
P_1	1	2	13
P_2	2	2	4
P_3	4	2	7
P_4	5	2	11
P_5	6	2	9

Prüfen Sie, ob ein gültiger Ablaufplan für die gegebenen Aufträge existiert!

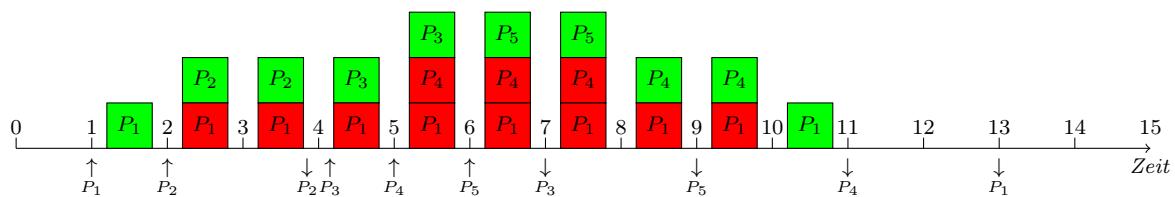
Lösung:

Scheduling gemäß *Earliest-Deadline-First (EDF)*

Prozess	Ankunftszeit	Rechenzeit	Deadline	Verweildauer	Wartezeit
P_1	1	2	13	10	8
P_2	2	2	4	2	0
P_3	4	2	7	2	0
P_4	5	2	11	5	3
P_5	6	2	9	2	0

$$\emptyset \text{ Wartezeit} = 2, 2$$

$$\emptyset \text{ Verweildauer} = 4, 2$$



Legende

- Prozess, rechnend (1 Zeiteinheit)

- Prozess, rechenbereit (1 Zeiteinheit)

\uparrow - Anfangszeit

\downarrow - Deadline

\Rightarrow Alle Deadlines können eingehalten werden. Dadurch ist ein gültiger Ablaufplan gegeben.

Aufgabe 2.3: Linux Scheduler

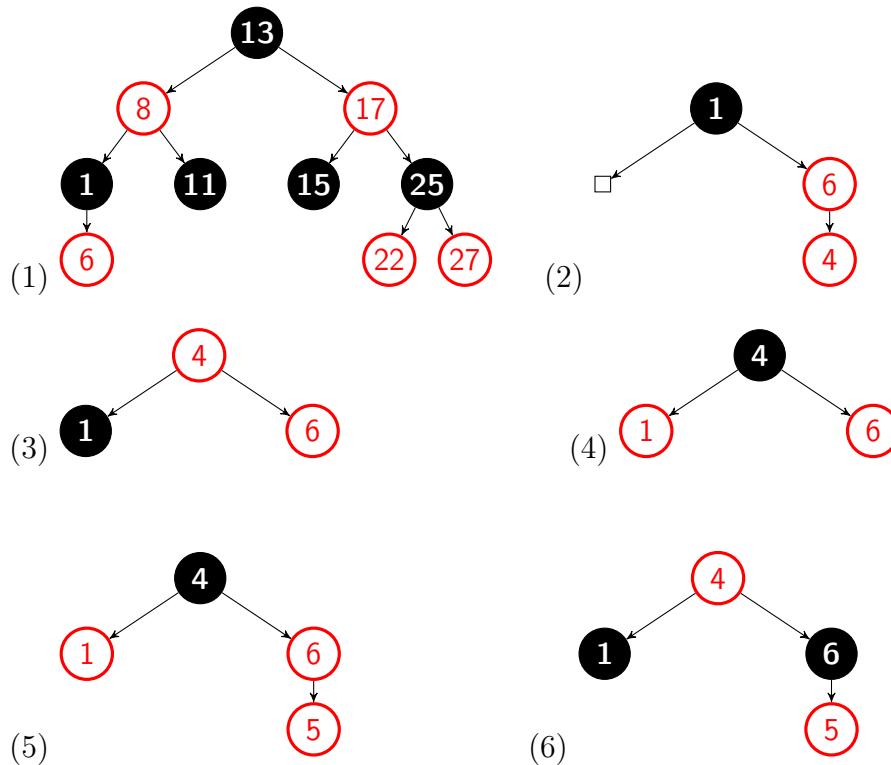
- a) Der Completely Fair Scheduler im aktuellen Linux benutzt als Datenstruktur für die Runqueue einen *Red-Black-Tree*.

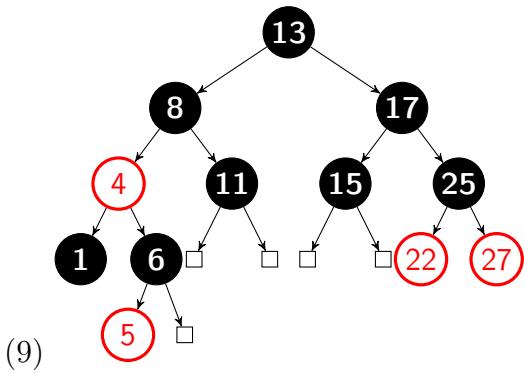
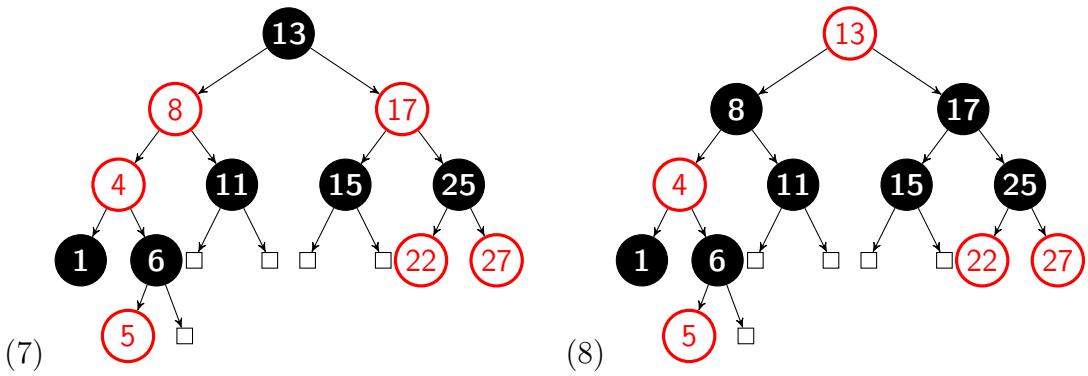
Geben Sie für den vorliegenden Red-Black-Tree (*siehe Abbildung 1*) alle Transformationsschritte an, die durchlaufen werden, wenn zunächst das Element mit dem Wert 4 und danach 5 eingefügt werden! Sie können sich dabei in der Darstellung auf den relevanten Teil-Baum beschränken.

- b) Geben Sie 2 Situationen an, in denen im CFS die `vruntime` gesetzt bzw. aktualisiert wird!

Lösung:

- a) Für einem Red-Black-Tree sind vier Regeln definiert.
1. Ein Knoten hat die Farbe rot oder schwarz.
 2. Die Wurzel eines Red-Black-Trees ist schwarz.
 3. Jeder rote Knoten darf nur schwarze Kindknoten besitzen.
 4. Ein neu angefügter Knoten ist immer rot.





1. Baum Vorgegebener Red-Black-Tree aus der Aufgabenstellung.
 2. Baum Wir betrachten nur noch den Teilbaum von der Eins ausgehend. Der neue rote Knoten 4 wird links an die rote 6 angefügt, das verstößt gegen unsere Regel 3, jeder rote Knoten darf nur schwarze Knoten besitzen.
 3. Baum Wir führen erst eine Linksrotation und dann eine Rechtsrotation durch, damit die 4 der Elternknoten von 1 und 6 ist.
 4. Baum Nun färben wir den Elternknoten 4 schwarz und den Kindknoten 1 schwarz. Daraus folgt, dass alle Voraussetzungen für einen Red-Black-Tree wiederhergestellt sind.
 5. Baum Wir fügen den neuen roten Knoten 5 links an den roten Knoten 6 an. Damit verletzen wir wieder die 3. Regel eines Red-Black-Trees.
 6. Baum Wir färben nun den Eltern- und Onkelknoten von 5 schwarz und den Großvaterknoten rot. Damit erhalten wir den 7.Baum.
 7. Baum Wieder wird die 3. Regel verletzt durch die Knoten 4 und 8.
 8. Baum Wir färben deswegen den Onkel- und Vaterknoten von 4 wieder schwarz und den Großvaterknoten rot.
 9. Baum Dadurch verletzen wir die 2.Regel eines Red-Black-Trees. Deswegen färben wir die Wurzel schwarz und erhalten somit den finalen Baum 9.

b) Die `vruntime` wird auf `minVruntime` gesetzt, wenn ein Task in den Red-Black-Tree kommt. `vruntime` wird bei periodischer Laufzeit erhöht.

Aufgabe 2.4: Windows Scheduler

- a) Gegeben sei ein Einprozessor-System mit Windows 2000. Weiterhin existieren 3 Prozesse (Base Prio 8), von denen zwei rechenbereit sind und einer (Prozess 2) auf das Netzwerk wartend blockiert ist. Prozess 2 erhält zum Zeitpunkt $t = 0$ die CPU, wenn das Netzwerk bereit ist.
- (i) Wie groß ist die Zeitscheibe von Prozess 2?
 - (ii) Nach wieviel Sekunden hat der Prozess 2 seine Basispriorität wieder erreicht?
 - (iii) Zeichnen Sie ein modifiziertes Gantt-Diagramm, aus dem die aktuellen Prioritäten der Prozesse ablesbar sind! Das Diagramm muss für jeden Prozess wenigstens 4 Zeitscheiben umfassen!
- b) Im zweiten Szenario ist Prozess 2 ein GUI-Thread. Beantworten Sie folgende Fragen für diesen Fall:
- (i) Wie groß ist die Zeitscheibe von Prozess 2?
 - (ii) Nach wieviel Sekunden hat der Prozess 2 seine Basispriorität wieder erreicht?
 - (iii) Zeichnen Sie ein modifiziertes Gantt-Diagramm, aus dem die aktuellen Prioritäten der Prozesse ablesbar sind! Das Diagramm muss für jeden Prozess wenigstens 4 Zeitscheiben umfassen!

Lösung:

a)	(i)	clock interrupt	0ms	10ms	20ms
		quantum value	6	3	0

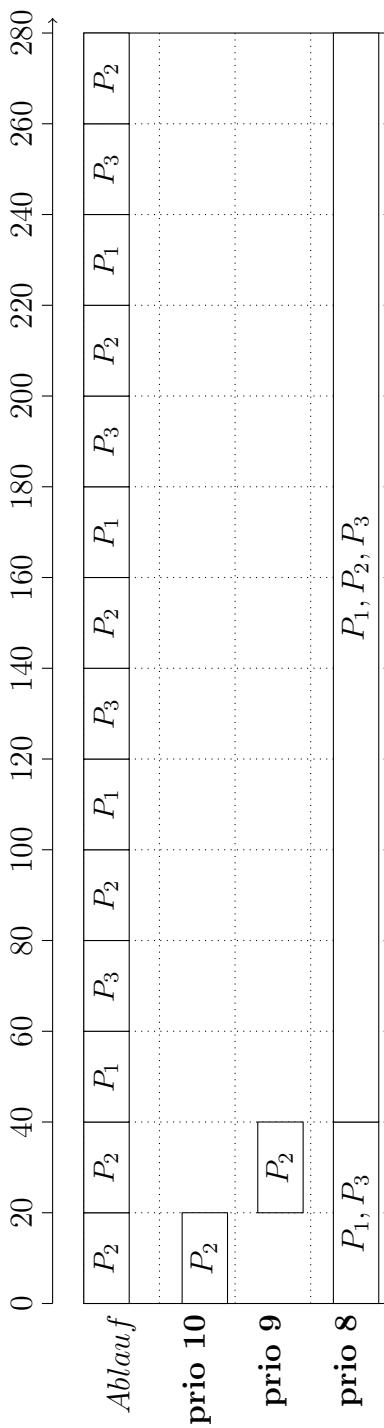
Aus der Tabelle folgt, dass die Zeitscheibe für den Prozess P_2 gleich 20ms beträgt.

- (ii) Alle drei Prozesse (P_1, P_2, P_3) besitzen eine Basispriorität von 8. P_2 wartet jedoch auf das Netzwerk, wodurch der Prozesse einen Networkboost von 2 bekommt. Aus der Vorlesung wissen wir, dass sich die *currentPrio* wie folgt berechnen lässt:

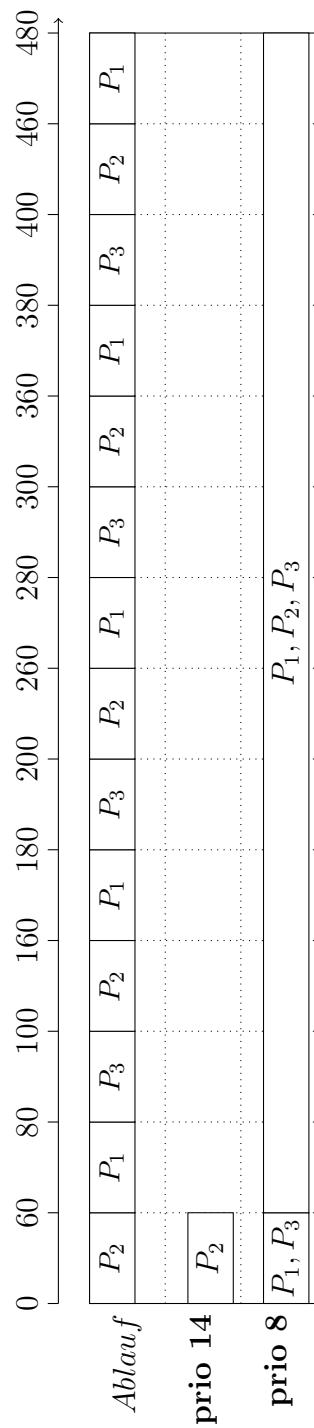
$$\text{currentPrio} = \min\{\text{basePrio} + \text{boost}, 15\}$$

Die neue *currentPrio* für P_2 ergibt sich also aus $\min\{8 + 2, 15\}$ und beträgt 10. Aus (i) wissen wir, dass eine Zeitscheibe 20ms beträgt. Die *currentPrio* von P_2 wird pro Zeitscheibe um 1 verringert. Heißt nach 20ms (1 Zeiteinheit) beträgt die *currentPrio* 9 und nach 40ms (2 Zeiteinheiten) beträgt die *currentPrio* wieder 8 und entspricht damit wieder der *basePrio*.

- (iii) siehe *Gantt-Diagramm 1*
- b) (i) Bei einem GUI-Thread ist die Zeitscheibe standardmäßig genau drei mal so lang wie die Zeitscheibe eines normales Threads(Vorlesungsfolie Seite 127). Daraus folgt also die Zeitscheibe für $P_2 : 3 \cdot 20ms = 60ms$.
- (ii) Da ein GUI Thread eine interaktive Sitzung ist, bekommt ein GUI-Thread einen Boost von 6. Daraus folgt für P_2 die $\text{currentPrio} = \min\{8 + 6, 15\} = 14$. Der Boost für eine interaktive Sitzung wird nur für eine Zeitscheibe gehalten, dass heißt, dass der Prozess P_2 nach 60ms wieder seine *basePrio* von 8 erreicht hat.
- (iii) siehe *Gantt-Diagramm 2*



Gantt-Diagramm 1



Gantt-Diagramm 2

Blatt 3

Aufgabe 3.1: IPC - Shared Memory

Schreiben Sie drei C-Programme (`master.c`), (`slave.c`) und (`freigabe.c`) gemäß nachfolgender Beschreibung!

- Das Master-Programm legt einen gemeinsamen Speicherbereich von 48 Byte an.
- Das Slave-Programm füllt den Speicher mit der Zeichenkette "Hier bin ich!".
- Das Masterprogramm gibt die eingegebenen Zeichen mit dem Kommentartext ("Wurde auch Zeit!") aus.
- Das dritte Programm soll nach Beendigung der beiden anderen Programme den gemeinsamen Speicher freigeben.

Beantworten Sie folgende Fragen:

- a) Starten Sie (`master.c`) und (`slave.c`). Was passiert, wenn Sie (`slave.c`) zuerst starten?
- b) Mit dem Kommando (`ipcs`) können Sie sich den gemeinsamen Speicher anzeigen lassen. Woran erkennen Sie, wieviele Prozesse den Speicher nutzen?
- c) Mit welchem Kommando können Sie die Speicherfreigabe umsetzen, wenn Ihnen dafür kein selbgeschriebenes Programm zur Verfügung steht?

(komplizierte) Lösung: In unserer Umsetzung erstellt ggf. `freigabe` den SHM, wenn es zuerst gestartet wird, damit der SHM von Anfang an überwacht werden kann. (siehe Umsetzung)

- a) `slave` beendet sich mit dem Fehler `shmget:No such file or directory`, da noch kein SHM erstellt wurde.
- b) Die Spalte `nattch` zeigt die Anzahl der verbundenen Prozesse zu einem Shared Memory Segment.

```
$ ipcs
----- Shared Memory Segments -----
key     shmid   owner    perms      bytes  nattch    status
0x00000000 262144  jakob    600        67108864    2        dest
0x00000000 360449  jakob    600        524288     2        dest
0x51060037 393218  jakob    600        10240      1
0x00000000 491523  jakob    600        524288     2        dest
```

- c) Mit dem Befehl `ipcrm` kann eine IPC Ressource entfernt werden. Zur Verwendung dieser Befehles stehen unterschiedliche Optionen zur Verfügung. Zum Beispiel entfernt `ipcrm -a` alle Ressourcen, `ipcrm -M shmkey` die Shared-Memory-Ressource mit dem genannten Schlüssel und `ipcrm -m shmid` die Shared-Memory-Ressource mit der genannten ID Nummer.

Aufgabe 3.2: Interprozesskommunikation

Geben Sie die Konsolenbefehle an, mit denen Sie folgende Aufgabenstellungen lösen:

- a) Geben Sie den Inhalt einer Datei (`textdatei.txt`) seitenweise aus!
- b) Zählen Sie die Zeilen einer Textdatei (`textdatei.txt`), in denen sowohl das Wort "Hallo" als auch das Wort "Nachbar" vorkommt!
- c) Geben Sie die Anzahl der laufenden `bash`-Instanzen aus!
- d) Geben Sie die ersten Zeile der Datei `textdatei.txt` auf dem Drucker aus!

Wieviele Prozesse werden durch die Konsolenbefehle in c) erzeugt? Welchen Mechanismus der Interprozesskommunikation nutzen diese Prozesse? Wie sind dabei Ein- und Ausgabe der beteiligten Prozesse gesetzt?

Lösung

- (a) `cat textdatei.txt | less`, dann kann man mit Leerzeichen zur nächsten Seite springen. Eine Realisierung der Aufgabe ist auch mit `more` möglich. Jedoch kann mit `more` nicht im Text zurück navigiert werden.
- (b) `cat textdatei.txt | grep 'Hallo \| Nachbar' | wc -l`
- (c) `ps -A | grep bash | wc -l`
Drei Prozesse werden erzeugt. Die Kommunikation findet über Pipelines statt. `ps -A` gibt die Prozesstabellen aus und gibt diese als Eingabe an `grep bash` weiter, welches nun alle Zeilen ausgibt, welche `bash` enthalten. Diese Ausgabe erhält nun `wc`, was mit der Option `-l` die Anzahl der Zeilen in der Konsole ausgibt, welche von `grep bash` geliefert wurden.
- (d) `cat textdatei.txt | head -n 1 | lpr`

Aufgabe 3.3: IPC - Signale

Programmieraufgabe

- (c) Der Prozess kann mit dem Konsolenbefehl `kill pid` beendet werden.

Aufgabe 3.4: Synchronisation: gemeinsame Variable

Betrachten wir die Prozesse P_1 und P_2 , welche die gemeinsamen Variablen x und y wie folgt aktualisieren:

Prozess P_1

```
/* fuehrt aus:
   x := x * y
   y ++
*/
LOAD R1, X
LOAD R2, Y
MUL R1, R2
STORE X, R1
INC R2
STORE Y, R2
```

Prozess B

```
/* fuehrt aus:
   x := x * y x++
   y := x * y
*/
LOAD R3, X
INC R3
LOAD R4, Y
MUL R4, R3
STORE X, R3
STORE Y, R4
```

Nehmen wir an, die Anfangsbelegung von x und y sind 2 und 3.

- Was ist das Ergebnis für x und y , wenn P_1 und P_2 in dieser Reihenfolge nacheinander ausgeführt wurden?
- Geben Sie einen verschachtelten Ablaufplan an, welcher das gleiche Resultat liefert!
- Geben Sie einen verschachtelten Ablaufplan an, welcher ein Resultat liefert, welches verschieden vom obigen ist!

Lösung:

(a)

Code	R1	R2	R3	R4	X	Y
/* P1 */	-	-	-	-	2	3
LOAD R1, X	2	-	-	-	2	3
LOAD R2, Y	2	3	-	-	2	3
MUL R1, R2	6	3	-	-	2	3
STORE X, R1	6	3	-	-	6	3
INC R2	6	4	-	-	6	3
STORE Y, R2	6	4	-	-	6	4
/* P2 */	6	4	-	-	6	4
LOAD R3, X	6	4	6	-	6	4
INC R3	6	4	7	-	6	4
LOAD R4, Y	6	4	7	4	6	4
MUL R4, R3	6	4	7	28	6	4
STORE X, R3	6	4	7	28	7	4
STORE Y, R4	6	4	7	28	7	28

Endergebnis: $X = 7, Y = 28$

(b)

Code	R1	R2	R3	R4	X	Y
/*Verschachtelt*/	-	-	-	-	2	3
LOAD R1, X	2	-	-	-	2	3
LOAD R2, Y	2	3	-	-	2	3
MUL R1, R2	6	3	-	-	2	3
STORE X, R1	6	3	-	-	6	3
LOAD R3, X	6	3	6	-	6	3
INC R3	6	3	7	-	6	3
INC R2	6	4	7	-	6	3
STORE Y, R2	6	4	7	-	6	4
LOAD R4, Y	6	4	7	4	6	4
MUL R4, R3	6	4	7	28	6	4
STORE X, R3	6	4	7	28	7	4
STORE Y, R4	6	4	7	28	7	28

Endergebnis: $X = 7, Y = 28$

(c)

Code	R1	R2	R3	R4	X	Y
/* verschachtelt verschieden */	-	-	-	-	2	3
LOAD R1, X	2	-	-	-	2	3
LOAD R3, X	2	-	2	-	2	3
LOAD R2, Y	2	3	2	-	2	3
INC R3	2	3	3	-	2	3
MUL R1, R2	6	3	3	-	2	3
LOAD R4, Y	6	3	3	3	2	3
STORE X, R1	6	3	3	3	6	3
MUL R4, R3	6	3	3	9	6	3
INC R2	6	4	3	9	6	3
STORE X, R3	6	4	3	9	3	3
STORE Y, R2	6	4	3	9	3	4
STORE Y, R4	6	4	3	9	3	9

Endergebnis: $X = 3, Y = 9$

Aufgabe 3.5: Banker's Algorithmus

Der in der Übung vorgestellte Banker's Algorithmus umgeht Verklemmungen, indem er nur sichere Betriebsmittelzustände zulässt.

Gegeben sei die folgende Situation:

- 4 Prozesse
- 3 Betriebsmittelklassen mit jeweils 5, 8 und 16 Instanzen

- Anforderungsmatrix: $Max = \begin{pmatrix} 4 & 1 & 4 \\ 3 & 1 & 4 \\ 3 & 7 & 13 \\ 1 & 1 & 6 \end{pmatrix}$

- Zuweisungsmatrix: $Allocation = \begin{pmatrix} 0 & 1 & 4 \\ 2 & 0 & 1 \\ 1 & 2 & 1 \\ 1 & 0 & 3 \end{pmatrix}$

- Ist der gegenwärtige Zustand des Systems sicher?
- Angenommen Prozess 1 fordert 1 Einheit des ersten Betriebsmittels an. Ist dieser neue Zustand sicher?
- Angenommen Prozess 3 fordert 6 Einheiten des dritten Betriebsmittels an. Ist dieser Zustand sicher?

Lösung:

- Die Need/Request Matrix ergibt sich aus Max – Allocation. Die Available Matrix berechnet sich, indem man von der Instanzenanzahl für jede Ressource die Anzahl der anfangs allokierten Instanzen für jede Ressource abzieht.
Insgesamt verfügbare Ressourcen := $(5 \ 8 \ 16)$

Available	Max	Allocation	Need/Request
$(1 \ 5 \ 7)$	$\begin{pmatrix} 4 & 1 & 4 \\ 3 & 1 & 4 \\ 3 & 7 & 13 \\ 1 & 1 & 6 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 4 \\ 2 & 0 & 1 \\ 1 & 2 & 1 \\ 1 & 0 & 3 \end{pmatrix}$	$\begin{pmatrix} 4 & 0 & 0 \\ 1 & 1 & 3 \\ 2 & 5 & 12 \\ 0 & 1 & 3 \end{pmatrix}$

P_1 muss warten da Request > Available

$(0 \ 4 \ 4)$	$\begin{pmatrix} 4 & 1 & 4 \\ 3 & 1 & 4 \\ 3 & 7 & 13 \\ 1 & 1 & 6 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 4 \\ 3 & 1 & 4 \\ 1 & 2 & 1 \\ 1 & 0 & 3 \end{pmatrix}$	$\begin{pmatrix} 4 & 0 & 0 \\ 0 & 0 & 0 \\ 2 & 5 & 12 \\ 0 & 1 & 3 \end{pmatrix}$
---------------	---	--	---

P_2 allokiert benötigte Ressourcen

$(3 \ 5 \ 8)$	$\begin{pmatrix} 4 & 1 & 4 \\ 3 & 1 & 4 \\ 3 & 7 & 13 \\ 1 & 1 & 6 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 4 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \\ 1 & 0 & 3 \end{pmatrix}$	$\begin{pmatrix} 4 & 0 & 0 \\ 0 & 0 & 0 \\ 2 & 5 & 12 \\ 0 & 1 & 3 \end{pmatrix}$
---------------	---	--	---

P_2 gibt Ressourcen wieder frei

P_3 muss warten da Request > Available

$(3 \ 4 \ 5)$	$\begin{pmatrix} 4 & 1 & 4 \\ 3 & 1 & 4 \\ 3 & 7 & 13 \\ 1 & 1 & 6 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 4 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \\ 1 & 1 & 6 \end{pmatrix}$	$\begin{pmatrix} 4 & 0 & 0 \\ 0 & 0 & 0 \\ 2 & 5 & 12 \\ 0 & 0 & 0 \end{pmatrix}$
---------------	---	--	---

P_4 allokiert benötigte Ressourcen

$(4 \ 5 \ 11)$	$\begin{pmatrix} 4 & 1 & 4 \\ 3 & 1 & 4 \\ 3 & 7 & 13 \\ 1 & 1 & 6 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 4 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \\ 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 4 & 0 & 0 \\ 0 & 0 & 0 \\ 2 & 5 & 12 \\ 0 & 0 & 0 \end{pmatrix}$
----------------	---	--	---

P_4 gibt Ressourcen wieder frei

$$(0 \ 5 \ 11) \quad \begin{pmatrix} 4 & 1 & 4 \\ 3 & 1 & 4 \\ 3 & 7 & 13 \\ 1 & 1 & 6 \end{pmatrix} \quad \begin{pmatrix} 4 & 1 & 4 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \\ 0 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 2 & 5 & 12 \\ 0 & 0 & 0 \end{pmatrix}$$

P_1 allokiert benötigte Ressourcen

$$(4 \ 6 \ 15) \quad \begin{pmatrix} 4 & 1 & 4 \\ 3 & 1 & 4 \\ 3 & 7 & 13 \\ 1 & 1 & 6 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \\ 0 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 2 & 5 & 12 \\ 0 & 0 & 0 \end{pmatrix}$$

P_1 gibt Ressourcen wieder frei

$$(2 \ 1 \ 3) \quad \begin{pmatrix} 4 & 1 & 4 \\ 3 & 1 & 4 \\ 3 & 7 & 13 \\ 1 & 1 & 6 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 3 & 7 & 13 \\ 0 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

P_3 allokiert benötigte Ressourcen

$$(5 \ 8 \ 16) \quad \begin{pmatrix} 4 & 1 & 4 \\ 3 & 1 & 4 \\ 3 & 7 & 13 \\ 1 & 1 & 6 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

P_3 gibt Ressourcen wieder frei

⇒ Da es eine realisierbare Abarbeitungsfolge für alle existierenden Prozesse gibt, gilt der Zustand als sicher. Die Abarbeitungsfolge ist P_2, P_4, P_1, P_3

- (b) Da der Prozess nun weniger Betriebsmittel fordert ist der Ablaufplan auf jeden Fall gesichert. Die Abarbeitung findet jetzt in der Reihenfolge P_1, P_2, P_3, P_4 statt. Die Ausgangsmatrizen ändern sich wie folgt:

$$\begin{array}{ccc} \text{Max} & \text{Allocation} & \text{Need/Request} \\ \begin{pmatrix} 1 & 1 & 4 \\ 3 & 1 & 4 \\ 3 & 7 & 13 \\ 1 & 1 & 6 \end{pmatrix} & \begin{pmatrix} 0 & 1 & 4 \\ 2 & 0 & 1 \\ 1 & 2 & 1 \\ 1 & 0 & 3 \end{pmatrix} & \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 3 \\ 2 & 5 & 12 \\ 0 & 1 & 3 \end{pmatrix} \end{array}$$

- (c) Das selbe gilt hier. Die Abarbeitung findet jetzt in der Reihenfolge P_2, P_3, P_4, P_1 statt. Die geänderten Ausgangsmatrizen sind:

$$\begin{array}{ccc} \text{Max} & \text{Allocation} & \text{Need/Request} \\ \begin{pmatrix} 4 & 1 & 4 \\ 3 & 1 & 4 \\ 3 & 7 & 6 \\ 1 & 1 & 6 \end{pmatrix} & \begin{pmatrix} 0 & 1 & 4 \\ 2 & 0 & 1 \\ 1 & 2 & 1 \\ 1 & 0 & 3 \end{pmatrix} & \begin{pmatrix} 4 & 0 & 0 \\ 1 & 1 & 3 \\ 2 & 5 & 5 \\ 0 & 1 & 3 \end{pmatrix} \end{array}$$

Blatt 4

Aufgabe 4.1: Fork - Threads

- a) Schreiben Sie ein C-Programm, das zunächst eine Datei `foo` öffnet und aus der Datei ein Zeichen liest und ausgibt, danach mittels `fork` einen Kindprozess erzeugt. Sowohl Elternprozess als auch Kindprozess sollen das nächste Zeichen der Datei lesen und mit der Angabe, wer gelesen hat, ausgeben. Schreiben Sie ein Programm, welches das umsetzt. Beantworten Sie folgende Fragen:
- i) Unter welcher Deskriptornummer wird die Datei `foo` in der Deskriptortabelle gelistet? Geben Sie die Antwort sowohl für den Eltern- als auch für den Kindprozess!
 - ii) Falls der Kindprozess eine weitere Datei `foobar` öffnet, unter welchen Deskriptornummern ist diese im Kind- bzw. Elternprozess zu sehen? Überprüfen Sie Ihre Vermutung durch Erweiterung des Programms!
- b) Verändern Sie das Programm aus a) so, dass statt `fork` ein neuer Thread erzeugt wird und die Ausgaben wie in a) erfolgen! Beantworten Sie die Fragen, wie sie in a) gestellt wurden!

Lösung:

- (a)
 - i) Die Datei `foo` ist in den Deskriptortabellen von Child und Parent unter der Deskriptornummer 3 gelistet.
 - ii) Die Datei `foobar` ist im Kindprozess unter der Deskriptornummer 4 gelistet. Im Elternprozess ist diese Datei nicht gelistet, da der Kindprozess einen separaten Speicherbereich besitzt und im Elternprozess nicht geöffnet wurde.
- b)
 - i) Die Datei `foo` ist unter Deskriptornummer 3 gelistet.
 - ii) Die Datei `foobar` ist sowohl im Hauptthread als auch im zweiten Thread unter Deskriptornummer 4 sichtbar, da beide auf dem gleichen Speicherbereich arbeiten.

Aufgabe 4.2: Interprozesskommunikation

Schreiben Sie ein C-Programm, das genau einmal

`"Wir sind Gruppe n!"` (*n* sei Ihre Gruppennummer)

an das Empfängerprogramm des Übungsleiters schickt! Das Programm des Übungsleiters liest TCP Nachrichten auf Host `gpu1.haiti.cs.uni-potsdam.de` (`141.89.59.177`) und Port `2018`. Zur Bestätigung eingegangener Nachrichten sendet es ein

`Daten erhalten (<Bytes>): <Empfangene Daten>`

zurück. Testen Sie Ihren Client zunächst mit einem eigenen Serverprogramm! Bitte geben Sie den Quellcode Ihrer Programme ab!

Aufgabe 4.3: Atomare Operationen

In der Vorlesung wurde der Maschinenbefehl CMPXCHG vorgestellt. Gegeben sei die Funktion `do_cmpxchg`, die analog zu `c_cmpxchg` aus der Vorlesung arbeitet, aber den von `CMPXCHG` gelesenen Wert zurückliefert, wenn sich dieser vom Vorgabewert unterscheidet.

Zur Generierung einer eindeutigen Zahl (ID) wird nun folgender Programmcode verwendet (Integer-Überläufe werden in Kauf genommen):

```
static unsigned long current_id = 0;

int generate_id() {
    return do_cmpxchg(&current_id, current_id + 1, current_id);
}
```

C-Wrapper für `cmpxchg` Anweisung:

```
int do_cmpxchg(int *ptr_dest, int new_value, int compare_value) {
    unsigned long tmp;

    /* LOCK CMPXCHG ueber Assembler aufrufen */

    /* tmp ist Rueckgabewert von CMPXCHG */
    return tmp;
}
```

Pseudo-C-Äquivalent zu `cmpxchg` aus der Vorlesung (ohne LOCK-Präfix nicht atomar):

```
int CMPXCHG(int * ptr_dest, int new_value, int compare_value) {

    /* read value at ptr_dest */
    int old_value = *ptr_dest;

    if (old_value == compare_value) {
        *ptr_dest = new_value;
        return new_value;
    } else {
        return old_value;
    }
}
```

Lösung: Da Threads parallel und im gleichen Speicherbereich arbeiten würden alle Threads die `generate_id()` gleichzeitig aufrufen den gleichen Wert zurückgeben.

Aufgabe 4.4: Synchronisation

Gegeben seien 3 gleichwertige Drucker mit einem gemeinsamen Verwaltungssystem:
Druckaufträge werden zwischengespeichert und von dem nächsten freien Drucker bearbeitet.
Welches grundlegende Problem der Synchronisation liegt vor? Entwickeln Sie eine Lösung für das Problem in Pseudocode basierend auf dem Semaphorkonzept.

Lösung:

Das Problem, welches vorliegt ist das Producer-Consumer-Problem.

```

int n                               /*n ist Buffergroesse*/
int next_w = 0;
int next_r = 0;
Semaphore mutex = 1      /*wechselseitiger Aushchluss*/
Semaphore free = n
Semaphore full = 0

/*Producer Funktion die zum Aufgeben eines Druckjobs aufgerufen wird.*/
addToQueue(jobitem) {
    DOWN(free);
    DOWN(mutex);
    write_item(jobitem, next_w);
    next_w = (next_w + 1) mod n;
    UP(mutex);
    UP(full);
}

/*Consumer Funktion, welche eine Druckerressource darstellt*/
printer() {
    while(True) {
        DOWN(full);
        DOWN(mutex);
        item = remove_item(next_r);
        next_r = (next_r + 1) mod n;
        UP(mutex);
        UP(free);
        print(item);
    }
}

/*Funktion die drei Printer erstellt*/
createPrinter() {
    for(int i = 0; i < 3; ++i) {
        createThread{printer()};
    }
}

```

Aufgabe 4.5: Synchronisation nebenläufiger Prozesse

Gegeben sei die Semaphorlösung für das Readers/Writers Problem aus der Vorlesung. (siehe 3.4.2)
 Im folgenden sei vorausgesetzt, dass jeder Prozess maximal drei Anweisungen (Zeilen) ausführen kann, bevor er unterbrochen wird. Dabei sollen Funktionsaufrufe vereinfacht durch ihren Quelltext ersetzt werden (falls vorhanden). Das Schedulingverfahren sei ein Round-Robin-Verfahren.
 Beschreiben Sie den Ablauf, wenn

- a) zwei Schreibprozesse W_1 und W_2 nebenläufig schreiben wollen!
- b) ein Prozess R_1 liest für 3 Zeitscheiben, während der dritten Zeitscheibe trifft ein Schreibprozess W_1 und danach wieder ein Leseprozess R_2 ein!
- c) ein Prozess W_1 schreibt für 2 Zeitscheiben, während der zweiten Zeitscheibe trifft ein Leseprozess R_1 und danach ein weiterer Schreibprozess W_2 ein!

Lösung: (*ohne c*)

`writecount` und `readcount` sind bei den Vergleichen mit `wc` und `rc` abgekürzt.

Initialwerte:

```
readcount = 0;
writecount = 0;
mutex_rc = 1;
mutex_wc = 1;
mutex_str = 1;
sem_w = 1;
sem_r = 1;
```

		Prozesse				Variablenbelegungen				
	W_1	W_2		rc	wc	mut_rc	mut_wc	mut_str	sem_w	sem_r
DOWN(mutex_wc); /*startwrite*/				0	0	1	1	1	1	1
writecount++;				0	0	1	0	1	1	1
if (wc==1):DOWN(sem_r);				0	1	1	0	1	1	1
DOWN(mutex_wc) /*startwrite*/ /				0	1	1	-1	1	1	0
UP(mutex_wc);				0	1	1	0	1	1	0
writecount++;				0	2	1	0	1	1	0
if (wc==1):DOWN(sem_r);				0	2	1	0	1	1	0
UP(mutex_wc);				0	2	1	1	1	1	0
DOWN(sem_w);				0	2	1	1	1	0	0
write(daten);				0	2	1	1	1	0	0
UP(sem_w); /*endwrite*/				0	2	1	1	1	1	0
DOWN(sem_w);				0	2	1	1	1	0	0
write(daten);				0	2	1	1	1	0	0
UP(sem_w); /*endwrite*/				0	2	1	1	1	1	0
DOWN(mutex_wc);				0	2	1	0	1	1	0
writecount-;				0	1	1	0	1	1	0
if (wc==0) :UP(sem_r);				0	1	1	0	1	1	0
DOWN(mutex_wc);				0	1	1	-1	1	1	0
UP(mutex_wc); /*fertig*/ /				0	1	1	0	1	1	0
writecount-;				0	0	1	0	1	1	0
if (wc==0):UP(sem_r);				0	0	1	0	1	1	1
UP(mutex_wc); /*fertig*/ /				0	0	1	1	1	1	1

a)

		Prozesse						Variablenbelegungen						
	R_1	W_1			R_2			rc	wc	mut_rc	mut_wc	mut_str	sem_w	sem_r
DOWN(mutex_str);								0	0	1	1	0	1	1
DOWN(sem_r);								0	0	1	1	0	1	0
DOWN(mutex_rc);								0	0	1	0	0	1	0
readcount++;								1	0	0	1	0	1	0
if ($rc==1$):DOWN(sem_w)								1	0	0	1	0	0	0
UP(mutex_rc);								1	0	1	1	0	0	0
UP(sem_r);								1	0	1	1	0	0	1
UP(mutex_str);								1	0	1	1	0	0	1
read(daten);								1	0	1	1	0	0	1
DOWN(mutex_wc);								1	0	1	0	1	0	1
writecount++;								1	1	0	1	0	1	1
if ($wc==1$):DOWN(sem_r);								1	1	0	1	0	1	0
DOWN(mutex_str);								1	1	0	0	0	0	0
DOWN(sem_r);								1	1	0	0	0	0	-1
DOWN(mutex_rc);								1	1	0	0	0	0	-1
readcount-;								0	1	0	0	0	0	-1
if ($rc==0$):UP(sem_w);								0	1	0	0	0	1	-1
UP(mutex_rc); /*fertig*/								0	1	0	1	0	1	-1
UP(sem_w);								0	1	1	1	0	1	-1
DOWN(mutex_wc);								0	1	1	0	0	0	-1
writecount-;								0	1	0	1	0	0	-1
if ($wc==0$):UP(sem_r);								0	1	0	1	0	0	-1
DOWN(mutex_rc);								0	0	0	0	0	0	0
readcount++;								1	0	0	0	0	1	0
if ($rc==1$):DOWN(sem_w)								1	0	0	0	0	0	0
UP(mutex_wc); /*fertig*/								1	0	0	1	0	0	0
UP(mutex_rc);								1	0	1	1	0	0	0
UP(sem_r);								1	0	1	1	1	0	1
UP(mutex_str);								1	0	1	1	1	0	1
read(daten);								1	0	1	1	1	0	1
DOWN(mutex_rc);								1	0	0	1	1	0	1
readcount-;								0	0	0	1	1	0	1
if ($rc==0$):UP(sem_w);								0	0	0	1	1	1	1
UP(mutex_rc);								0	0	1	1	1	1	1
/*fertig*/								0	0	1	1	1	1	1

b)

Blatt 5

Aufgabe 5.1: Synchronisation von Threads

Schreiben Sie ein C-Programm, das nebenläufig 2 neue Threads erzeugt, die jeder n -mal die Ausgabe "Ich bin Prozeß meine_PID, ThreadID: meine_ThreadID!" auf den Bildschirm schreiben und nach jeder Ausgabe 1 Sekunde schlafen! **Die Ausgabe soll streng alternierend erfolgen!**

n soll als Kommandozeilenargument übergeben werden. Diese Threads sollen außer- dem am Ende ausgeben: "Mein Vater hat die PID Vater-PID und mein Bruder hat die Thread-ID Bruder-TID!"

Um Busy Waiting zu vermeiden, sollen Mutexe und/oder Bedingungsvariable benutzt werden!

Aufgabe 5.2: POSIX-Threads, Reader/Writer Locks

Im folgenden finden Sie eine Implementierung eines Synchronisierungsmechanismus für mehrere Schreib- und Lesethreads. Die Implementierung basiert auf einem Mutex und einer Condition Variable. Ein lesender Thread ruft zunächst die Funktion `pthread_rdwr_rlock` auf, führt seinen Lesezugriff durch und schließt den Vorgang mit `pthread_rdwr_runlock` ab. Analog dazu arbeitet der Schreibprozess mit den Funktionen `pthread_rdwr_wlock` und `pthread_rdwr_wunlock`.

Beantworten Sie folgende Fragen für ein Einprozessorsystem, wenn $n > 1$ Lese- und $k > 1$ Schreibthreads aktiv sind und eine korrekte Initialisierung vorausgesetzt werden kann

1. Wie viele Threads können sich zur gleichen Zeit in der while -Schleife von `pthread_rdwr_rlock` befinden?
2. Wie viele Schreibprozesse können gleichzeitig mit der Ausführung von `pthread_rdwr_wunlock` beginnen?
3. Gegeben sei ein Thread R_1 , der `pthread_rdwr_rlock` bereits ausgeführt hat. Es treffen nun ein weiterer Lesethread R_2 und ein Schreibthread W_1 ein. Welcher von beiden erhält im weiteren Verlauf zuerst Zugriff auf das Lock? Welcher abnormale Prozess/Threadzustand kann in Folge dieses Effekts auftreten?

```
typedef struct {
    int readers_reading;
    int writer_writing;
    pthread_mutex_t mutex;
    pthread_cond_t lock_free;
} pthread_rdwr_t;

int pthread_rdwr_init(pthread_rdwr_t *rdwrf, pthread_rdwrattr_t *attrp) {
    rdwrf->readers_reading = 0;
    rdwrf->writer_writing = 0;
    pthread_mutex_init(&(rdwrf->mutex), NULL);
    pthread_cond_init(&(rdwrf->lock_free), NULL);
```

```

        return 0;
    }

int pthread_rdwr_rlock(pthread_rdwr_t *rdwrf) {
    pthread_mutex_lock(&(rdwrf->mutex));

    while(rdwrf->writer_writing) {
        pthread_cond_wait(&(rdwrf->lock_free), &(rdwrf->mutex));
    }

    rdwrf->readers_reading++;
    pthread_mutex_unlock(&(rdwrf->mutex));

    return 0;
}

int pthread_rdwr_wlock(pthread_rdwr_t *rdwrf) {
    pthread_mutex_lock(&(rdwrf->mutex));

    while (rdwrf->writer_writing || rdwrf->readers_reading) {
        pthread_cond_wait(&(rdwrf->lock_free), &(rdwrf->mutex));
    }

    rdwrf->writer_writing++;
    pthread_mutex_unlock(&(rdwrf->mutex));

    return 0;
}

int pthread_rdwr_runlock(pthread_rdwr_t *rdwrf) {
    pthread_mutex_lock(&(rdwrf->mutex));

    /* sanity check: if no one acquired the lock, release mutex and return
     * error */
    if (rdwrf->readers_reading == 0) {
        pthread_mutex_unlock(&(rdwrf->mutex));

        return -1;
    } else {
        rdwrf->readers_reading--;
        if (rdwrf->readers_reading == 0) {
            pthread_cond_signal(&(rdwrf->lock_free));
        }
        pthread_mutex_unlock(&(rdwrf->mutex));
    }

    return 0;
}

int pthread_rdwr_wunlock(pthread_rdwr_t *rdwrf) {
    pthread_mutex_lock(&(rdwrf->mutex));

    /* sanity check again */
    if (rdwrf->writer_writing == 0) {

```

```

    pthread_mutex_unlock(&(rdwrf->mutex));

    return -1;
} else {
    rdwrf->writer_writing = 0;

    pthread_cond_broadcast(&(rdwrf->lock_free));
    pthread_mutex_unlock(&(rdwrf->mutex));

    return 0;
}
}

```

Begründen Sie jeweils Ihre Antworten!

Lösung:

- Der Aufruf der Schleife ist zwar mit einem Mutex abgesichert, jedoch wird das Mutex `pthread_mutex_t mutex` mit dem Aufruf von `pthread_cond_wait` wieder freigegeben und der Thread blockiert in der Schleife. Dieser Aufruf findet statt, wenn gerade ein Schreibthread aktiv ist. So können beliebig viele Threads auf die angegebene Bedingung `pthread_cond_t lock_free` warten. Dies bedeutet, dass beliebig viele Lesethreads auf die Beendigung eines Schreibthreads in der `while`-Schleife warten können.
- Der Funktionskontext ist mit dem `pthread_mutex_t mutex` abgesichert, welches zur Beginn der Ausführung mit `pthread_mutex_lock` akquiriert wird. Somit kann immer nur ein Schreibthread den Funktionsinhalt ausführen. Es wird aber ohnehin immer nur ein Schreibthread existieren, da ein Schreibthread erst gestartet werden kann, wenn kein Lese- oder Schreibthread mehr existiert (siehe int `pthread_rdwr_wlock` Zeilen 39-41):

```

while (rdwrf->writer_writing || rdwrf->readers_reading) {
    pthread_cond_wait(&(rdwrf->lock_free), &(rdwrf->mutex));
}

```

- R_2 wird zuerst Zugriff auf das Lock erhalten, da dieser Lesethread bei folgender Abfrage nicht auf die Beendigung eines Schreibthreads warten muss:

```
while(rdwrf->writer_writing)
```

Somit kann es beliebig viele Lesethreads geben, die gleichzeitig lesen. Da bereits der Lesethread R_1 existiert kann der Schreibthread das Lock nicht zuerst erhalten, da er auf die Beendigung aller Lesethreads warten muss (*siehe 2.*).

In diesem Beispiel könnte nur ein abnormale Zustand auftreten, wenn ein Lesethread auf Daten wartet, welche erst von dem Schreibthread W_1 geschrieben werden. Im Allgemeinen könnte ein abnormale Zustand auftreten, wenn immer wieder neue Lesethreads eintreffen und der Schreibthread somit nie mit der Ausführung beginnen kann.

Aufgabe 5.3: Speicherverwaltung

Ein Minicomputer mit 256K Speicher benutzt das Buddy-System in der Speicherverwaltung. Wenn nacheinander Speicherbereiche von 19K, 4K, 12K, 29K und 55K angefordert werden, wieviele Blöcke sind dann noch übrig und wie sind ihre Größen und Adressen? Wie ändern sich diese Größen, wenn die Blöcke von 4K und 12K Größe wieder freigegeben werden? Wie groß ist dann der größte verfügbare Speicherbereich, wieviel Speicher steht insgesamt noch zur Verfügung?

Lösung:

Mit Hilfe des Buddy-Systems findet auf dem 256K-Speicher schrittweise die Allokation von 19K, 4K, 12K, 29K und 55K Speicherblöcken statt (*Abbildungen a bis e*). Nachdem alle Speicherbereiche angefordert und die Blöcke geteilt und zugewiesen wurden sind noch 4 Speicherblöcke frei (*Abbildung e*):

4K an Adresse 36

8K an Adresse 40

32K an Adresse 96

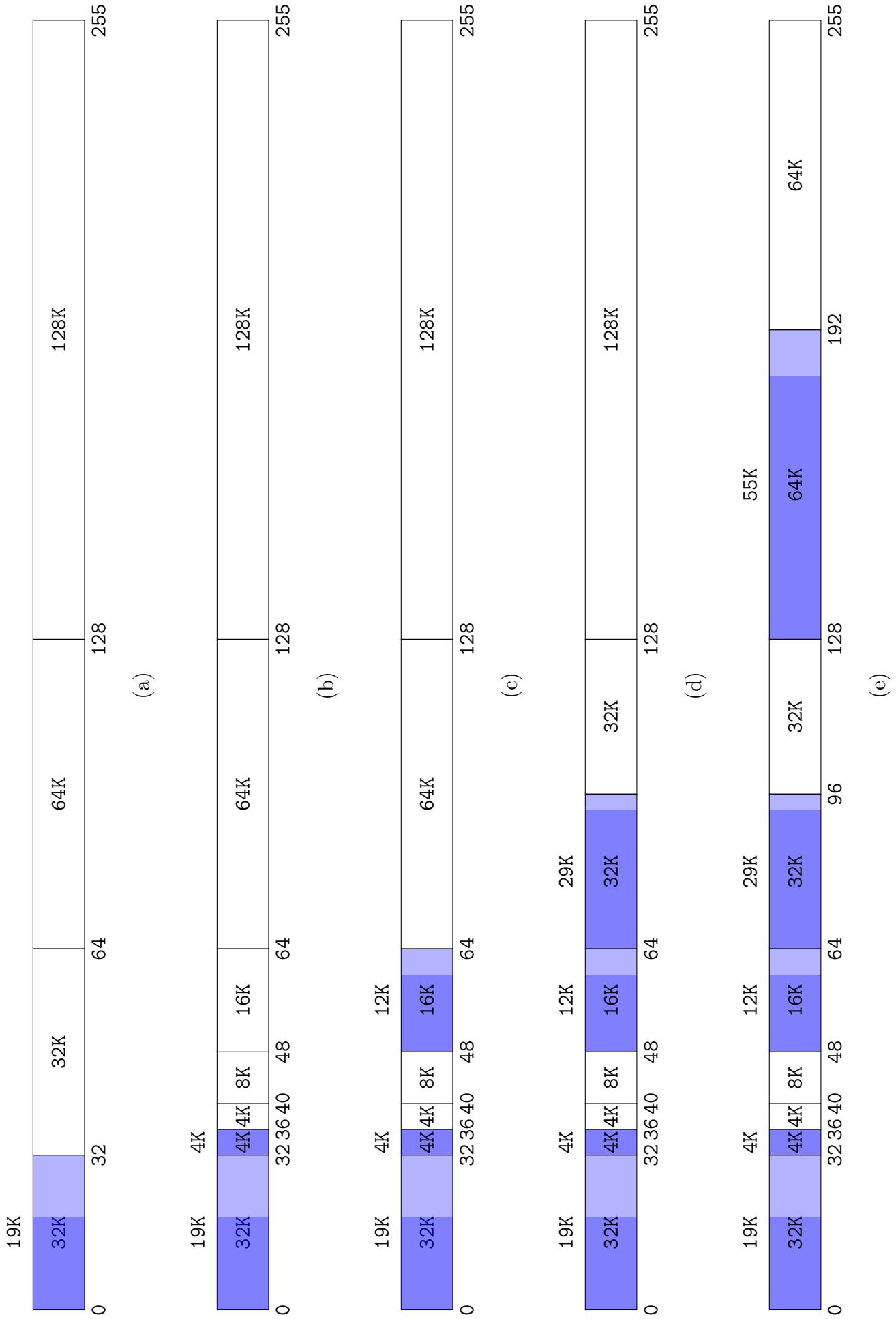
64K an Adresse 192

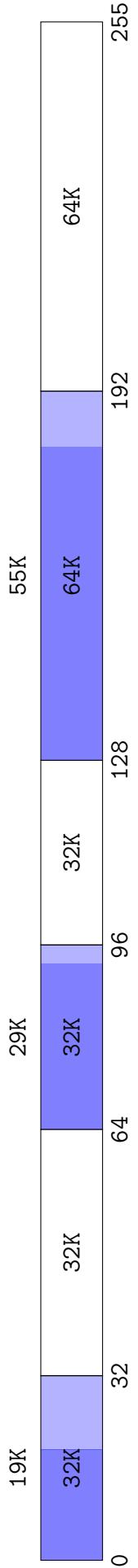
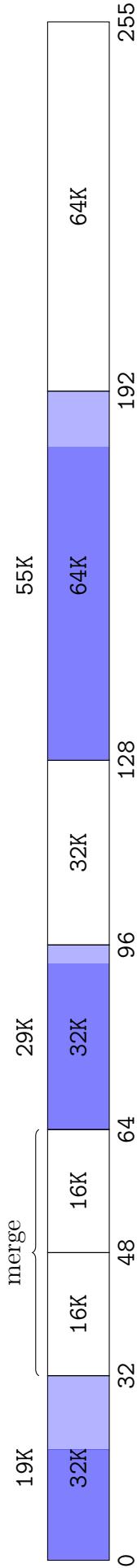
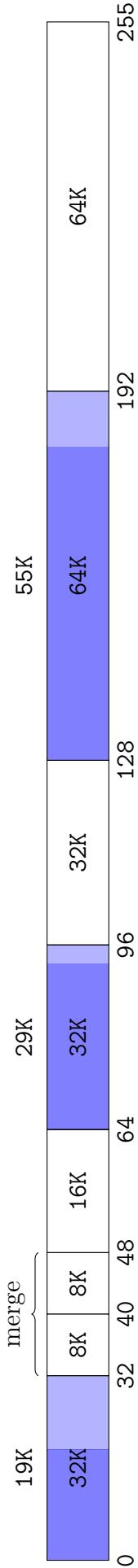
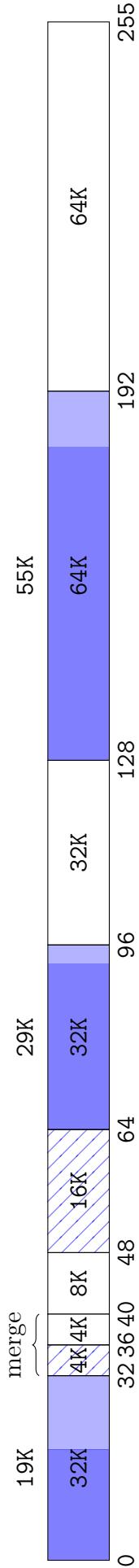
Im folgenden werden die Blöcke mit den Speicherbereichen der Größen 4K und 12K freigegeben (*Abbildung f*). Nun können die Speicherblöcke im Adressbereich 32-63 zu einem Block zusammengeführt werden. Die Vereinigung findet gemäß der Abbildungen (*f*) bis (*i*) statt. Nach der Zusammenführung sind 3 freie Speicherblöcke mit einer Gesamtgröße von 128K vorhanden, wobei der größte Block eine Größe von 64K besitzt:

32K an Adresse 32

32K an Adresse 96

64K an Adresse 192





Aufgabe 5.4: Speicherverwaltung

Zeigen Sie, dass LRU ein gutartiger Seitenersetzungsalgorithmus ist!

Lösung:

Definition 26: "Ein Paging Algorithmus heißt gutartig, falls

$$\forall W \in \mathbb{N}^{\mathbb{N}} \quad \forall m \in \mathbb{N} : F(A, m + 1, W) \leq F(A, m, W)$$

Vergleich der Seitenfehler für eine Hauptspeichergröße m mit der Anzahl der Seitenfehler bei einer Hauptspeichergröße von $m + 1$ für $m = 3$.

		Seitenzugriffe W											
		0	1	2	3	0	1	4	0	1	2	3	4
Speicher- plätze m	1	0	0	0	3	3	3	4	4	4	2	2	2
	2		1	1	1	0	0	0	0	0	0	3	3
	3		2	2	2	1	1	1	1	1	1	1	4
LRU- Liste	1	0	1	2	3	0	1	4	0	1	2	3	4
	2		0	1	2	3	0	1	4	0	1	2	3
	3		0	1	2	3	0	1	4	0	1	2	
SF	*	*	*	*	*	*	*	*	*	*	*	*	*

$$F(\text{LRU}, 3, W) = \sum \text{SF} = 10$$

		Seitenzugriffe W											
		0	1	2	3	0	1	4	0	1	2	3	4
Speicher- plätze m	1	0	0	0	0	0	0	0	0	0	0	0	4
	2		1	1	1	1	1	1	1	1	1	1	1
	3		2	2	2	2	4	4	4	4	3	3	
	4		3	3	3	3	3	3	3	2	2	2	
	5	*	*	*	*	*	*	*	*	*	*	*	*
LRU- Liste	1	0	1	2	3	0	1	4	0	1	2	3	4
	2		0	1	2	3	0	1	4	0	1	2	3
	3		0	1	2	3	0	1	4	0	1	2	
	4		0	1	2	3	3	3	4	0	1		
	5	*	*	*	*	*	*	*	*	*	*	*	*
SF	*	*	*	*	*	*	*	*	*	*	*	*	*

$$F(\text{LRU}, 4, W) = \sum \text{SF} = 8$$

Da $F(\text{LRU}, 4, W) \leq F(\text{LRU}, 3, W)$ für dieses Beispiel gilt, kann der LRU-Algorithmus als gutartig angesehen werden.

Aufgabe 5.5: Speicherverwaltung

Die Behandlung eines Seitenfehlers erfordert i. A. Maßnahmen zur Ersetzung einer Seite in einem Seitenrahmen (Kachel) des Arbeitsspeichers. Dazu muss zunächst eine andere Seite aus dem Arbeitsspeicher verdrängt werden.

Bei der Ausführung eines Benutzerprogramms tritt folgende Sequenz von Seitenzugriffen auf:

2 7 6 7 1 4 7 2 5 5 1

Es steht ein physikalischer Adressraum mit 4 Seitenrahmen zur Verfügung. Arbeiten Sie die angegebene Seitenzugriffsfolge gemäß der Seitenersetzungsstrategien:

- i) BO (Belady's optimaler Algorithmus),
- ii) FIFO (First-In-First-Out),
- iii) LRU (Least-Recently-Used),
- iv) Auslagern einer Seite, die nicht zum Working-Set gehört (Working-Set-Parameter $T = 3$).

ab, und vergleichen Sie diese anhand der Seitenfehleranzahl!

Lösung:

		Seitenzugriffe W										
		2	7	6	7	1	4	7	2	5	5	1
i)	Speicherplätze m	1	2	2	2	2	2	2	2	5	5	5
		2	7	7	7	7	7	7	7	7	7	7
		3		6	6	6	4	4	4	4	4	4
		4				1	1	1	1	1	1	1
	$d(m, t)$	1	7	6	5	4	3	2	1	∞	1	∞
		2	2	1	3	2	1	∞	∞	∞	∞	∞
		3		∞								
		4				6	5	4	3	2	1	∞
		SF	*	*	*	*	*	*	*			

$$\sum \text{SF} = 6$$

		Seitenzugriffe W										
		2	7	6	7	1	4	7	2	5	5	1
ii)	Speicherplätze m	1	2	2	2	2	4	4	4	4	4	4
		2	7	7	7	7	7	7	2	2	2	2
		3		6	6	6	6	6	6	5	5	5
		4				1	1	1	1	1	1	1
	FIFO-Liste	1	2	2	2	2	7	7	6	1	1	1
		2	7	7	7	7	6	6	1	4	4	4
		3		6	6	6	1	1	4	2	2	2
		4				1	4	4	2	5	5	5
		SF	*	*	*	*	*	*	*	*		

$$\sum \text{SF} = 7$$

		Seitenzugriffe W										
		2	7	6	7	1	4	7	2	5	5	1
iii)	Speicherplätze m	1	2	2	2	2	4	4	4	4	4	1
		2		7	7	7	7	7	7	7	7	7
		3		6	6	6	6	6	2	2	2	2
		4				1	1	1	1	5	5	5
	LRU-Liste	1	2	7	6	7	1	4	7	2	5	5
		2		2	7	6	7	1	4	7	2	5
		3		2	2	6	7	1	4	7	7	2
		4			2	6	6	1	4	4	4	7
SF		*	*	*	*	*	*	*	*	*	*	*

$$\sum \text{SF} = 8$$

		Seitenzugriffe W										
		2	7	6	7	1	4	7	2	5	5	1
iv)	Speicherplätze m	1	2	2	2	2	4	4	4	4	4	1
		2		7	7	7	7	7	7	7	7	7
		3		6	6	6	6	6	2	2	2	2
		4			1	1	1	1	5	5	5	5
	Working-Set		2	2	2	6	1	1	1	2	2	2
			7	6	7	6	4	4	4	5	5	5
			7	7	7	7	7	7	7	7	7	7
		SF	*	*	*	*	*	*	*	*	*	*

$$\sum \text{SF} = 8$$

BO liefert für die gegebene Folge von Seitenzugriffen wie erwartet mit 6 Seitenfehlern das effizienteste Seitenmanagement. Sowohl LRU als auch WS weisen 8 Seitenfehler auf. Somit ist der $FIFO$ Algorithmus in diesem Beispiel der beste Algorithmus der sich realistisch implementieren lässt.

Blatt 6

Aufgabe 6.1: POSIX-Threads, verkettete Liste, Master-Worker-Parallelisierungskonzept

seeehr lange Programmieraufgabe

Aufgabe 6.2: Verwaltung virtueller Adressraum

Programmieraufgabe

Aufgabe 6.3: Dateisystem

UNIX kennt die Zugriffsrechte `rwx` für Dateien. Gegeben sei ein System mit den folgenden Nutzern und drei Gruppen: Ken und Barbie (Gruppe Projektleitung), Bob und Alice (Gruppe Sekretariat) sowie Max, Moritz und Philipp (Gruppe Mitarbeiter)

Falls möglich: setzen Sie die Rechte so, dass die nachfolgenden Forderungen erfüllt sind!
Geben Sie jeweils Besitzer und Besitzgruppe der Datei an!

- (a) Max, Moritz und Philipp möchten auf ihre Datei `entwurf.svg` lesend und schreibend Zugriff haben, aber sonst soll diese Datei niemand lesen können.
- (b) Zusätzlich soll ihr Projektleiter Ken auf besagte Datei `entwurf.svg` Leserechte erhalten.
- (c) Ken und Moritz möchten die vertrauliche Datei `gehalt.txt` gemeinsam bearbeiten. Niemand sonst, insbesondere nicht Philipp, soll darauf Zugriff haben.
- (d) Für die Datei `Projektbericht` sollen alle Projektleiter und alle Mitglieder der Gruppe Sekretariat Lese- und Schreibrechte haben.

Windows NT unterscheidet hingegen für jede Datei die Rechte:

`Full Control - Modify - Read & Execute - Read - Write`

und für Verzeichnisse zusätzlich noch das Recht: `List Folder Contents`.

Die Rechte können individuell sowie für Gruppen vergeben werden. Lösen Sie die Aufgaben (a) bis (d) für Windows NT mit Hilfe von Access Control Lists (ACL) und geben Sie diese für jeden Teilaufgabe an!

Lösung:

1. Unix:

				owner	group	file
a)	-	<code>rw-</code>	<code>rw-</code>	--	Max	Mitarbeiter
b)	-	<code>r-</code>	<code>rw-</code>	--	Ken	Mitarbeiter
c)						Funktioniert nicht ohne neue Gruppe
d)	-	<code>rw-</code>	--	<code>rw-</code>	Ken	Mitarbeiter
						Projektbericht

2. Windows NT:

file	user/group	rights
a) entwurf.svg	Mitarbeiter	Read, Write
b)	Ken	Read
c) gehalt.txt	Ken Moritz	Read, Write Read, Write
d) Projektbericht	Projektleiter Sekretariat	Read, Write Read, Write

Aufgabe 6.4: Dateisystem

Unter Unix wird eine Datei durch einen Inode repräsentiert. Gehen Sie jetzt davon aus, dass jeder Inode 12 Zeiger auf direkte Blöcke und je einen Zeiger auf einen Einfach-, Doppelt- und Dreifach-Indirekt-Block enthält. Die Größe eines Systemblocks bzw. eines Plattensektors ist 1024 Byte. Ein Pointer auf einen Plattenblock ist 32 Bit groß.

Wenn der File-Inode sich bereits im Hauptspeicher befindet: Wieviele Plattenzugriffe sind erforderlich, um auf ein Byte an der Position 4747720 zuzugreifen?

Lösung:

$$\frac{\text{Blockgröße } 1024 \text{ Byte}}{\text{Pointergröße } 4 \text{ Byte}} = 256 \text{ Pointer pro Block}$$

Anzahl

12	direkter Block	$12 \cdot 1$	=	12
1	einfach indirekter Block	$1 \cdot 256$	=	256
1	zweifach indirekter Block	$1 \cdot 256^2$	=	65912
1	dreifach indirekter Block	$1 \cdot 256^3$	=	16777216

Es sind 4 Plattenzugriffe notwendig da das 4747720 Byte zwischen 65912 und 16777216 liegt und somit auf die File-Inode und dann auf den dreifach indirekten Block zugegriffen werden muss.

Aufgabe 6.5: I/O-Scheduling

Plattenanfragen treffen beim Plattentreiber für folgende Zylinder ein: 9, 2, 5, 47, 28, 19, 29 (in dieser Reihenfolge). Eine Bewegung des Plattenarmes dauert 1 Millisekunde pro Zylinder.

Wie lange ist die Suchzeit (seek time) bei:

1. FIFO,
2. Shortest-Seek-First,
3. Scan-/Elevator-Algorithmus, bei dem sich der Plattenarm zunächst aufwärts bewegen soll?

In allen drei Fällen befindet sich der Plattenarm zunächst bei Zylinder 31!

Lösung:

1. FIFO (First In First Out)

Reihenfolge	9	2	5	47	28	19	29	Summe
Bewegungen	22	7	3	42	19	9	10	112

Daraus folgt die Suchzeit = 112ms.

2. Shortest Seek First:

Reihenfolge	29	28	19	9	5	2	47	Summe
Bewegungen	2	1	9	10	4	3	45	74

Daraus folgt die Suchzeit = 74ms.

3. Scan-/Elevator-Algorithmus:

Reihenfolge	47	29	28	19	9	5	2	Summe
Bewegungen	16	18	1	9	10	4	3	61

Daraus folgt die Suchzeit = 61ms.

Blatt 7

Aufgabe 7.1: Fehlererkennung

- a) Die folgende Bitfolge soll über ein Netzwerk übertragen werden:

1011 0110

Ermitteln Sie die Bitfolge, die unter Benutzung des Hammingcodes übertragen wird!

- b) Sie empfangen die folgende Bitfolge, die den Hammingcode benutzt:

1100 0110 0110

Gab es einen Übertragungsfehler? Wenn ja, an welcher Bitposition im Original trat der Fehler auf? (Achtung, die Bitfolgen von a) und b) haben nichts miteinander zu tun.)

Lösung:

a)

binär Index	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100
Codewort Index	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}
Paritätsbitstellen	2^0	2^1		2^2				2^3				
Bittyp	p_0	p_1	d_1	p_2	d_2	d_3	d_4	p_3	d_5	d_6	d_7	d_8
Bitwert	1	1	1	0	0	1	1	0	0	1	1	0

Das Paritätsbit p_i wird also über alle Stellen c_j des Codeworts berechnet, in denen an der i -ten Stelle der Binärkodierung des Index j eine logische Eins steht:

$$c_1 = p_0 = c_3 \oplus c_5 \oplus c_7 \oplus c_9 \oplus c_{11}$$

$$= 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1$$

$$= 1$$

$$c_2 = p_1 = c_3 \oplus c_6 \oplus c_7 \oplus c_{10} \oplus c_{11}$$

$$= 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1$$

$$= 1$$

$$c_4 = p_2 = c_5 \oplus c_6 \oplus c_7 \oplus c_{12}$$

$$= 0 \oplus 1 \oplus 1 \oplus 0$$

$$= 0$$

$$c_8 = p_3 = c_{10} \oplus c_{11} \oplus c_{12}$$

$$= 1 \oplus 1 \oplus 0$$

$$= 0$$

- b) Um die Übertragenen Daten auf Fehler zu prüfen werden alle Paritätsbits neu berechnet. Falls die neu Berechneten Paritätsbits mit den übermittelten übereinstimmen liegt kein Fehler vor. Wenn sich die neu berechneten Werte von den übertragenen unterscheiden, dann wird der Index i der fehlerhaften Paritätsbitstellen c_i addiert. Diese Summe identifiziert die Indexstelle des fehlerhaften Bits.

binär Index	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100
Codewort Index	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}
Paritätsbitstellen	2^0	2^1		2^2				2^3				
Bittyp	p_0	p_1	d_1	p_2	d_2	d_3	d_4	p_3	d_5	d_6	d_7	d_8
Bitwert	1	1	0	0	0	1	1	0	0	1	1	0

$$\begin{aligned} c_1 &= p_0 = c_3 \oplus c_5 \oplus c_7 \oplus c_9 \oplus c_{11} \\ &= 0 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \\ &= 0 \end{aligned}$$

$$\begin{aligned} c_2 &= p_1 = c_3 \oplus c_6 \oplus c_7 \oplus c_{10} \oplus c_{11} \\ &= 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \\ &= 0 \end{aligned}$$

$$\begin{aligned} c_4 &= p_2 = c_5 \oplus c_6 \oplus c_7 \oplus c_{12} \\ &= 0 \oplus 1 \oplus 1 \oplus 0 \\ &= 0 \end{aligned}$$

$$\begin{aligned} c_8 &= p_3 = c_{10} \oplus c_{11} \oplus c_{12} \\ &= 1 \oplus 1 \oplus 0 \\ &= 0 \end{aligned}$$

Aufgabe 7.2: Rechnernetze: CRC

Es soll die Bitfolge 0111 0101 1110 0001 als Nutzdaten übertragen werden. Als Generatorpolynom werde $G(x) = x^8 + x^7 + x + 1$ verwendet. Welche Bitfolge wird gesendet, wenn das Verfahren Bitstopfen verwendet? Geben Sie Ihren Lösungsweg an! Überprüfen Sie auf Empfängerseite die Richtigkeit der übertragenen Daten! (Machen Sie die Probe!)

Lösung:

Sender:

Bitfolge: 0111 0101 1110 0001

Generatorpolynom $G(x) = x^8 + x^7 + x + 1 = 110000011$ mit dem Grad = 8.

Die Bitfolge wird nun durch das Generatorpolynom dividiert, damit man einen Rest erhält. Dafür wird ausschließlich das exklusive OR(\oplus) verwendet, wobei gilt $1 \oplus 1 = 0$; $0 \oplus 1 = 1$; $1 \oplus 0 = 1$; $0 \oplus 0 = 0$.

Dadurch folgt:

Der Rest setzt sich durch die 8 ersten Bits der unteren Bitfolge zusammen, da der Grad = 8 ist. Somit erhalten wir den

Rest = 00010110.

Die gesendeten Daten setzen sich jetzt aus der Bitfolge + Rest zusammen:

0111 0101 1110 0001 0001 0110

Jeder Rahmen beginnt und endet mit einem speziellen Bitmuster (01111110), um festzulegen wo die gesendeten Daten anfangen und wo sie enden, wodurch die gesendete Bitfolge:

0111 1110 0111 0101 1110 0001 0001 0110 0111 1110

entsteht. Wenn in der Bitfolge die Bitsequenz 11111 auftauchen würde, dann würde nach dieser Sequenz eine zusätzliche 0 eingefügt werden. Bei der obigen Bitfolge ist Bitstopfen nicht nötig.

Empfänger:

Der Empfänger überprüft die empfangenen Daten, indem die Bitfolge ohne Anfang und Ende wieder durch das Generatorpolynom dividiert wird. Wenn bei der Division 0 rauskommt, dann wurden die Daten richtig übertragen.

$$\begin{array}{r}
 0\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 1\\
 1\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 1 \\
 \hline
 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 0 \\
 1\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 1 \\
 \hline
 0\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 0 \\
 1\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 1 \\
 \hline
 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1 \\
 1\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 1 \\
 \hline
 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0 \\
 1\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 1\ 0 \\
 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0 \\
 1\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0 \\
 0\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0 \\
 1\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0 \\
 0\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0 \\
 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0 \\
 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 1\ 0
 \end{array}$$

Aus der Division der Bitfolge mit dem Generatorpolynom folgt, dass die Daten die vom Sender gesendet wurden richtig sind.

Aufgabe 7.3: Rechnernetze: Leitungsauslastung

Gegeben sei eine Satellitenverbindung über eine Entfernung von 50 000 km mit Übertragungsrate 1 Mbps, Rahmengröße 1000 bit und Ausbreitungsgeschwindigkeit $2 \cdot 10^8 \text{ m/s}$. Bestimmen Sie die Leitungsauslastung für

1. ein Stop-and-Wait-Protokoll (ohne Fehler),
 2. ein Schiebefensterprotokoll mit 7-Bit Sendefolgenummern und Fenstergröße 127 (ohne Fehler).

Aufgabe 7.4: Sicherungsschicht

- a) Welche Form der Flusskontrolle gibt es in HDLC (*High-level Data Link Control*) und welche in Ethernet? Bei welchem Protokoll kann es trotzdem auf Schicht 2 zum Pufferüberlauf kommen?
 - b) Wieso kann für PPPoE das Feld Datenlänge im Ethernethrahmen in "Typ" ümdefiniert werden?

Lösung:

- a) HDLC verwendet zur Flusskontrolle ein Schiebefenster-Protokoll. Bei diesem kann der Sender nur eine maximale Anzahl von 8 Datenrahmen verschicken, ohne das der Empfänger den Erhalt der Datenrahmen mit einer Quittung bestätigt.

Teil III

Klausuren

Klausur 2010 1

Aufgabe 1.1:

- Kann es auf einem Einprozessorsystem parallele Prozesse geben? Begründen Sie Ihre Antwort!
- Erklären Sie den Begriff Nebenläufigkeit und beschreiben Sie eine Vorgehensweise im Betriebssystem!
- Gegeben war das C-Programm `myfork.c`

```
int main(void) {
    pid_t child_pid = fork();
    printf("The result is: %d\n", child_pid);
    return EXIT_SUCCESS;
}
```

Wie sieht die Ausgabe beim Ausführen des kompilierenden Programms aus, wenn während der Ausführung folgende Informationen durch den ps-Befehl zur Verfügung stehen?

S	UID	PID	PPID	TTY	TIME	CMD
S	2060	13069	11139	pts/1	00:00:00	myfork
S	2060	13070	13069	pts/1	00:00:00	myfork

Lösung:

- Nein, da auf einem Einprozessorsystem Prozesse nur nebenläufig oder sequentiell rechnen können und ein Prozessor nur einen Befehl "gleichzeitig" ausführen kann.
- Nebenläufigkeit bedeutet, dass sich die Laufzeiten von Prozessen überschneiden.
Definition aus der Vorlesung:
Prozess A und B mit den Laufzeiten (a_1, a_2) und (b_1, b_2) heißen nebenläufig $\Leftrightarrow [a_1, a_2] \cap [b_1, b_2] \neq \emptyset$
Der Scheduler im Betriebssystem kümmert sich um die Verteilung der Rechenzeit.
- `The result is: 13070`
`The result is: 0`

Aufgabe 1.2:

Gegeben war folgende Tabelle:

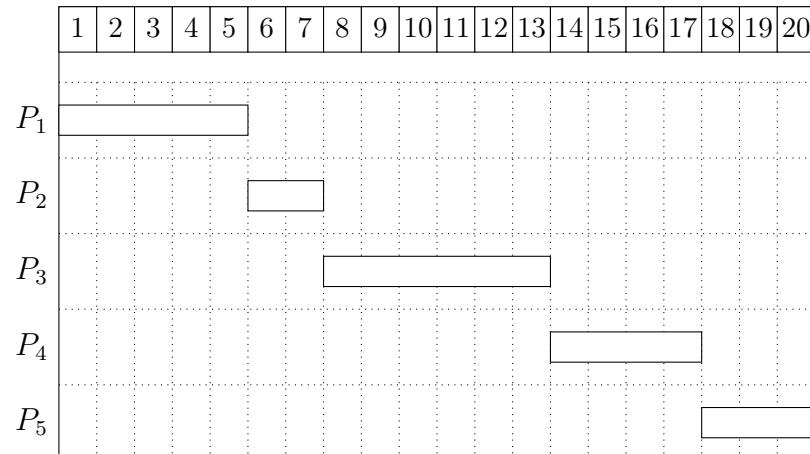
Prozess	Ankunft	Rechenzeit
P1	0	5
P2	0	2
P3	2	6
P4	3	4
P5	3	3

Es sollten 3 Gantt-Diagramme zu den Strategien FCFS, SJF und RoundRobin (Zeitschritt=1) gezeichnet werden.

Es war jeweils auch die mittlere Verweilzeit und mittlere Wartezeit anzugeben.

Lösung:

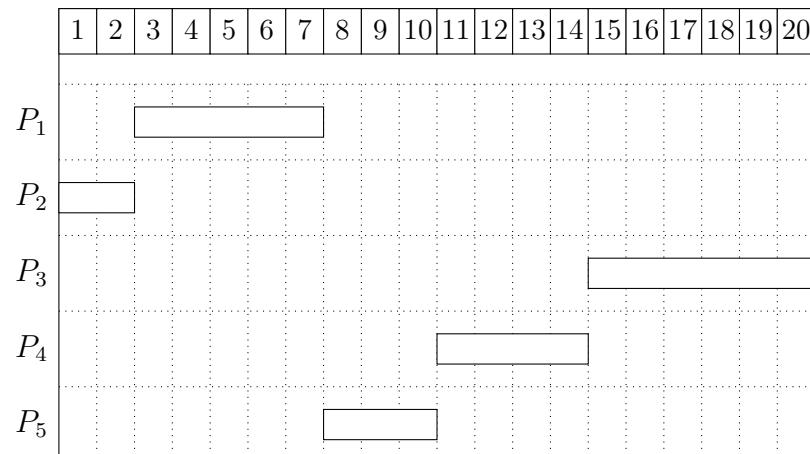
- FIFO:



P_i	Ankunft	Rechenzeit	Wartezeit	Verweilzeit
P_1	0	5	0	5
P_2	0	2	5	7
P_3	2	6	5	11
P_4	3	4	10	14
P_5	3	3	14	17
\emptyset			6,8	10,8

hier: Verweilzeit = Antwortzeit, da keine Unterbrechungen vorliegen

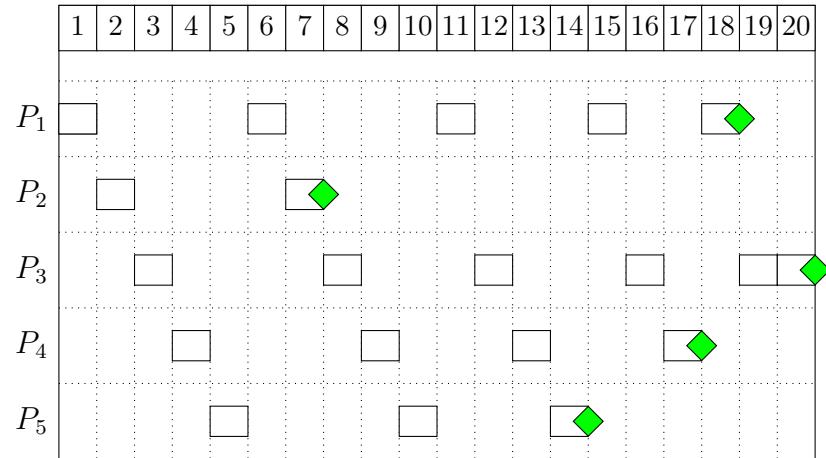
- SJF:



P_i	Ankunft	Rechenzeit	Wartezeit	Verweilzeit
P_1	0	5	2	7
P_2	0	2	0	2
P_3	2	6	12	18
P_4	3	4	7	11
P_5	3	3	4	7
\emptyset			5	8

hier: Verweilzeit = Antwortzeit, da keine Unterbrechungen vorliegen

- Round-Robin



P_i	Ankunft	Rechenzeit	Wartezeit	Verweilzeit	Antwortzeit
P_1	0	5	0	5	18
P_2	0	2	1	3	7
P_3	2	6	0	6	18
P_4	3	4	0	4	14
P_5	3	3	1	4	11
\emptyset			0,4	4,4	13,4

Aufgabe 1.3:

1. Nennen Sie einen Vorteil von Pipes gegenüber Signalen und einen Nachteil von Pipes gegenüber Sockets.
2. Welche Art der IPC (*Interprozesskommunikation*) würden sie für folgende Szenarien empfehlen (und Begründung):
 - (a) Datenaustausch Mehrprozessmaschine zwischen einem Prozess und zwei von ihm erzeugten Threads
 - (b) Aufwecken eines schlafenden Prozesses

Lösung:

- Für die Kommunikation mit Signalen steht nur eine begrenzte Anzahl an Signalen bereit. Bei Pipes hingegen können beliebige Daten übertragen werden.

Für die Verwendung von Pipes wird ein gemeinsamer Adressraum (gemeinsamer Speicher) benötigt, für Sockets nicht.

- (a) Eigentlich keine IPC nötig, Pipes
 (b) Ein schlafender Prozess kann mit Signalen geweckt werden

Aufgabe 1.4:

In einem Rechensystem mit 4 Seitenrahmen (anfangs leer) weise ein Prozess die folgende Sequenz von Seitenzugriffen auf:

$$3, 2, 4, 1, 1, 4, 7, 3, 5, 1, 4, 2, 3, 1$$

- Es sollten die Seitenersetzungsstrategien Belady's optimaler Algorithmus (BO) und FIFO angewendet werden und die Anzahl der Seitenfehler verglichen werden.
- Ist BO praktisch in einem Betriebssystem möglich? (Begründen!)

Lösung:

- Belady's Algorithmus:

	Seitenzugriffe W														
	3	2	4	1	1	4	7	3	5	1	4	2	3	1	
Speicherplätze m	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
	2	2	2	2	2	2	7	7	5	5	5	2	2	2	2
	3	4	4	4	4	4	4	4	4	4	4	4	4	4	4
	4	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$d(m, t)$	1	7	6	5	4	3	2	1	5	4	3	2	1	∞	∞
	2	10	9	8	7	6	∞								
	3	3	2	1	5	4	3	2	1	∞	∞	∞	∞	∞	∞
	4	1	5	4	3	2	1	4	3	2	1	∞			
SF	*	*	*	*	*	*	*	*	*	*	*	*	*	*	

First in First out:

	Seitenzugriffe W														
	3	2	4	1	1	4	7	3	5	1	4	2	3	1	
Speicherplätze m	3	3	3	3	3	3	7	7	7	7	7	2	2	2	2
	2	2	2	2	2	2	2	3	3	3	3	3	3	3	1
	3	4	4	4	4	4	4	4	5	5	5	5	5	5	5
	4	1	1	1	1	1	1	1	1	1	4	4	4	4	4
FIFO	1	3	2	4	1	1	1	7	3	5	5	4	2	2	1
	2	3	2	4	4	4	1	7	3	3	5	4	4	2	
	3	3	2	2	2	4	1	7	7	3	5	5	4		
	4	3	3	3	2	4	1	1	7	3	3	3	5		
SF	*	*	*	*	*	*	*	*	*	*	*	*			

2. Nein, da die Seitenanforderungen im Vorraus nicht bekannt sind

Aufgabe 1.5:

Es soll die Bitfolge 1100 1111 1001 als Nutzdaten übertragen werden. Als Generatorpolynom werde $x^4 + x + 1$ verwendet.

1. Welche Bitfolge wird gesendet?
2. Warum wird diese Bitfolge tatsächlich nicht so gesendet, warum ist eine Anpassung nötig?
Geben Sie den tatsächlich gesendeten HDLC-Frame an.
3. Geben Sie je ein Beispiel für ein Protokoll der Sicherungsschicht an!
 - (a) Peer-zu-Peer Verbindung wird unterstützt
 - (b) ist ein Broadcast-Protokoll.

Lösung:

1. Bitfolge: 1100 1111 1001

Generatorpolynom $G(x) = x^4 + x + 1 = 10011$ mit dem Grad = 4.

Die Bitfolge wird nun durch das Generatorpolynom dividiert, damit man einen Rest erhält.
Dafür wird ausschließlich das exklusive OR(\oplus) verwendet, wobei gilt $1 \oplus 1 = 0$; $0 \oplus 1 = 1$;
 $1 \oplus 0 = 1$; $0 \oplus 0 = 0$.

Dadurch folgt:

$$\begin{array}{r}
 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
 1 & 0 & 0 & 1 & 1 \\
 \hline
 0 & 1 & 0 & 1 & 0 & 1 \\
 1 & 0 & 0 & 1 & 1 \\
 \hline
 0 & 0 & 1 & 1 & 0 & 1 & 1 \\
 1 & 0 & 0 & 1 & 1 \\
 \hline
 0 & 1 & 0 & 0 & 0 & 1 \\
 1 & 0 & 0 & 1 & 1 \\
 \hline
 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
 1 & 0 & 0 & 1 & 1 \\
 \hline
 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
 1 & 0 & 0 & 1 & 1 \\
 \hline
 0 & 0 & 0 & 1 & 1 & 0
 \end{array}$$

Der Rest setzt sich durch die 4 ersten Bits der unteren Bitfolge zusammen, da der Grad = 4 ist. Somit erhalten wir den

$$\text{Rest} = 0110.$$

Die gesendeten Daten setzen sich jetzt aus der Bitfolge + Rest zusammen:

$$1100 \ 1111 \ 1001 \ 0000 \ 0110$$

Jeder Rahmen beginnt und endet mit einem speziellen Bitmuster (01111110). Somit erhalten wir das Bitmuster:

01111110 1100 1111 1001 0000 0110 01111110

2. Sobald die Sicherungsschicht des Senders 5 aufeinanderfolgende Einsen im Datenstrom entdeckt, füllt sie den abzusendenen Bitstrom automatisch mit einer Null auf. Dies wird Bitstopfen genannt. Dadurch erhalten wir den tatsächlich gesendeten Rahmen:

01111110 11001 1111**0** 01000 00110 01111110

3.

Aufgabe 1.6:

Szenario eines Wohnhauses mit Aufzug. Es dürfen nur maximal 3 Personen gleichzeitig im Fahrstuhl sein. Es gibt die Aktionen:

- `fahrstuhl_anfordern(Bewohner b)`
- `in_fahrstuhl_einsteigen(Bewohner b)`
- `etage_aussuchen(Bewohner b)`
- `in_etage_fahren(Bewohner b)`
- `aus_fahrstuhl_aussteigen(Bewohner b)`

1. Es war eine Semaphorlösung gefragt, die sicherstellt, dass sich maximal 3 Personen im Fahrstuhl befinden.
2. Nun sollte zusätzlich zu den Bedingungen aus a) auch noch sichergestellt werden, dass immer nur eine Person durch die schmale Tür geht.
3. Der Fahrstuhl muss auch mal gewartet werden. Dann befindet sich der Monteur alleine im Fahrstuhl und es muss sichergestellt werden, dass sich kein Fahrgäst im Fahrstuhl befindet. Der Monteur muss nicht bevorzugt behandelt werden. Für den Monteur soll es noch die Aktion `warte_fahrstuhl(Monteur m)` geben.

Lösung:

1. Producer-Consumer Problem:

```
// Bedingung, dass es nur eine Etage gibt
// b) die Tuer ist abgesichert durch den Semaphor fahrstuhl

semaphor mitfahrer = 3 /* max 3 Personen */
semaphor fahrstuhl = 1 /* Mutex */

/* Producer Einsteigen */
einstiegen(Bewohner b) {
    fahrstuhl_anfordern(b)
    DOWN (fahrstuhl)
    DOWN (mitfahrer)
```

```

        in_Fahrstuhl_einsteigen(b)
        etage_aussuchen(b)
        UP(Fahrstuhl)
    }

/* Consumer Aussteigen */
aussteigen(Bewohner b) {
    DOWN(Fahrstuhl)
    aus_fahrstuhl_aussteigen(b)
    UP(mitfahrer)
    UP(fahrstuhl)
}

2. b & c)

// Bedingung, dass es nur eine Etage gibt
// b) die Tuer ist abgesichert durch den Semaphor fahrstuhl

semaphor mitfahrer = 3 /* max 3 Personen */
semaphor fahrstuhl = 1 /* Mutex */

/* Producer Einsteigen */
einsteigen(Bewohner b) {
    fahrstuhl_anfordern(b)
    DOWN(fahrstuhl)
    DOWN(mitfahrer)
        in_Fahrstuhl_einsteigen(b)
        etage_aussuchen(b)
    UP(Fahrstuhl)
}

/* Consumer Aussteigen */
aussteigen(Bewohner b) {
    DOWN(Fahrstuhl)
        aus_fahrstuhl_aussteigen(b)
    UP(mitfahrer)
    UP(fahrstuhl)
}

// Verzichten auf Mutex, da Monteur nur in dieser Funktion taetig wird.
FahrstuhlWarten(Monteur m) {
    fahrstuhl_anfordern(m)
    DOWN(fahrstuhl)
    DOWN(mitfahrer) /* 3x zur Sicherstellung, dass */
    DOWN(mitfahrer) /* keine Mitfahrer im Fahrstuhl */
    DOWN(mitfahrer) /* sich noch befinden. */
        in_Fahrstuhl_einsteigen(m)
        etage_aussuchen(m)
        warte_Fahrstuhl(m)
    UP(Fahrstuhl)
}

```

INDEX Stichwortverzeichnis 1

176

Symbols

4.3 BSD UNIX Scheduler 18

A

Abnormales

 Prozessverhalten 10

Atomare Operationen .. 148

B

Banker's Algorithmus .. 144

Belady's Anomalie 56

Belady's optimaler
Algorithmus 56, 160

Berkeley Fast File System

 63

Betriebssystem 5

Bitstopfen 117

Buddy-Verfahren..... 54

Burstfehler..... 120

Busy Waiting 34

C

cmpxchg 37

Completely Fair Scheduler
 19

CRC 119, 166

D

Dateisysteme..... 59

Dateitypen..... 59

Deadlock 10, 134

Demand-Paging..... 55

Descriptor 25

Descriptor-Tabelle..... 25

Deskriptornummer 147

Deskriptortabelle..... 147

Dispatcher-Komponente . 17

Dynamische Prioritäten . 18

E

Earliest-Deadline-First . 136

Echtzeit-Scheduling 136

Elevator-Algorithmus .. 163

Erweiterte Access Control
Lists 68

F

Fehlererkennung 165

First-In-First-Out ... 15, 56,
 160, 163

Fork 134, 147

G

Generatorpolynom 166

gutartig 56

gutartiger Seitenerset-
zungsalgorithmus
 159

H

Hamming-Code 119

Hammingcode..... 165

I

i-node 63, 163

I/O-Scheduling..... 163

indirekter Block..... 163

Interprozesskommunikation
 23, 142, 147, 171

Shared Memory 141

Signale..... 142

K

Kernel-Mode 7

kritischer Bereich 34

L

Laufzeit 6

 nebenläufig..... 6

 parallel..... 6

 sequentiell..... 6

Least-Recently-Used 56, 160

Linux Dateisystem 64

Livelock..... 10

Lokalitätsprinzip 55, 57

M

Master-Worker-Modell .. 23

Memory Mapped Files .. 61
Monitor 42
Mutual Exclusion 34

N

nicht gutartig 56

P

Paging-Verfahren..... 55

Parallelisierungskonzepte 23

Peterson's Algorithmus.. 35

Pipeline..... 23

Pipes..... 25, 171

Plattenanfragen..... 163

Plattenlaufwerk 59

Pool-of-Thread..... 23

Priority Boosting..... 18

Producer-Consumer

 Problem..... 39

Protection Domain 67

Prozess 6, 9

 -zustände..... 10, 133

 beenden..... 12

 erzeugen 11

 kontrolle 23

Pthread Mutex..... 43

R

Race Condition 34

Readers-Writers Problem 40

Rechenzeit 14

Red-Black-Tree 20

Red-Black-Tree (Linux
Scheduler) 137

reentrant 22

Ressourcen 5

Round-Robin..... 18

Runqueue-Verwaltung... 18

S

Scheduler-Komponente .. 17

Scheduling 13

Schiebefenster-Protokolle

 121

Seitenfehler 55

Semaphor	38	Nachteile	55	V	
Nachteile	42			Vergissfilter	18 f
Shortest Job First	16			Verweilzeit	14
Shortest-Seek-First	163				
Signale	23			W	
Speicherverwaltung	54, 159			Wartezeit	14
Starvation	10			Windows 2000: NTFS ..	66
Starvation Free	34			Windows Scheduler	139
Statische/Dynamische Sei- tenersetzungsv erfahren	57			Working-Set-Modell	57
Stop-and-Wait-Protokolle	120				
Superblock	63				
Swapping	54				
				Zeitscheibenlänge	18
				Zugriffsrechte	162