INSTITUT FÜR INFORMATIK
DER LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

**Projektarbeit**

# Strukturelle Überarbeitung des Aufgaben-Fundus der Vorlesung Informatik 3

## Entwurf neuer Übungsaufgaben & Implementierung einheitlicher Konzepte zur Archivierung, Versionierung, Erweiterung und Vorschau

Thomas Schaaf

Aufgabensteller: Prof. Dr. Hans Jürgen Ohlbach
Betreuer: Bernhard Lorenz
Abgabetermin: 25. Februar 2005

# Inhaltsverzeichnis

# 1 Aufgabenstellung & Zielsetzung

## 1.1 Über die Vorlesung Informatik 3

Die Vorlesung Informatik 3 ist Teil des Pflicht-Programms im Informatik-Grundstudium an der LMU München. Sie findet zur Zeit in jedem Wintersemester als vierstündige Vorlesung statt, ergänzt um zweistündige Präsenzübungen (Tutorien) in kleinen Gruppen. Die Vorlesung richtet sich an Studenten der Informatik im dritten Fachsemester. Ihr Schwerpunkt liegt auf der Darstellung der Konzepte moderner Betriebssysteme. Im Vorlesungsverzeichnis heißt es:

> Typische Aufgaben der Systemprogrammierung sind die Programmierung des Betriebssystems sowie von Dienstleistungsprogrammen, wie beispielsweise Editoren, Compiler, Interpreter. Die Vorlesung gibt einen Überblick über die wesentlichen Aufgaben und Problembereiche eines Betriebssystems, wobei insbesondere auf die Bereiche Synchronisation, Prozess-Kommunikation, Verwaltung des Rechnerkerns sowie auf den Bereich der Speicherverwaltung eingegangen wird. Zur praktischen Umsetzung der in der Vorlesung eingeführten Konzepte wird als Programmiersprache in den Übungen Java (insbesondere die Thread-API) eingesetzt.

## 1.2 Übungsbetrieb

### 1.2.1 Allgemeiner Ablauf des Übungsbetriebs

Im Laufe des Semesters werden zwischen 12 und 14 Übungsblätter ausgegeben (pro Woche eines), mit denen der Vorlesungsstoff anhand praktischer Aufgaben vertieft wird. Jedes Übungsblatt wird in einem zweistündigen Tutorium besprochen. Die aktive Teilnahme am Übungsbetrieb ist Voraussetzung für die Zulassung zur Abschlussklausur am Semesterende. Auf jeden Übungsblatt sind einige Aufgaben als Hausaufgaben gekennzeichnet. Diese müssen von den Studenten selbständig gelöst und wöchentlich abgegeben werden. An drei zufällig ausgewählten und vorher nicht bekannten Terminen werden die Hausaufgaben korrigiert und bewertet. Jeder Student, der in mindestens zwei der drei korrigierten Hausaufgaben mindestens 50 Prozent der Maximalpunktzahl erreicht, wird zur Abschlussklausur zugelassen.

### 1.2.2 Erstellung der Übungsblätter

Da die Vorlesung abwechselnd von unterschiedlichen Dozenten gehalten wird, sind auch Gliederung, Inhalte und Literaturquellen manchmal sehr verschieden. Aktuelle Entwicklungen aus dem Bereich der Betriebssysteme (z.B. neue Dateisysteme) fließen soweit wie möglich in die Übungen ein. Daher werden alle Übungsblätter in jedem Semester neu erstellt, wobei selbstverständlich zu einem gewissen Anteil auf Aufgaben früherer Semester zurück gegriffen wird. Jedes Übungsblatt ist prinzipiell eine **Zusammenstellung** dieser Art:

- Vorlesungsparameter (Semester, Dozent, Abgabetermin usw.)
- Hinweise und aktuelle Informationen
- die eigentlichen Aufgaben

Ein Übungsblatt in seiner unkompilierten Version ist allerdings nicht ein einziges Text-Dokument, das alle Informationen enthält, sondern besteht aus mehreren Dateien. Am wichtigsten: die Quell-Dateien der einzelnen Aufgaben. Jede Aufgabe liegt in Form eines LaTeX-Quelldokuments vor. Das ermöglicht es, mehrere Aufgaben durch Referenzierung zu einem Übungsblatt zusammenzufassen. Diese Referenzierung erfolgt für jedes Übungsblatt in dessen Body-File („body.tex“), das etwa so aussieht:

```
\input{_VorlesungsParameter}
\nummerUebungsblatt{1} \date{20.10.2004}
\nummerErsteFrage{1}

\Ankuendigungen{Die Anmeldung zu den Übungen ist freigeschaltet.}
\zuLesen{Skript, Kapitel 1}
%\zuWiederholen{}
% ----------------------------------------------------------------
\begin{document}
\maketitle

\input{H_BetriebssystemeGrundlagen}
\input{T_VergleichBetriebssysteme}
\input{T_UnixBefehle}

\end{document}
```

Das Repository ist der „Behälter“ für alle jemals erstellten Übungsaufgaben zu dieser Vorlesung. Wie es strukturiert ist, und warum das zunehmend zu Problemen führte, wird in Kürze erläutert.

### 1.3 Diese Projektarbeit

Das Wissen um den Ablauf des Übungsbetriebs bildet die Grundlage für diese Projektarbeit. Aufgabe und Zielsetzung dieser Arbeit ist die vollständige **strukturelle, thematische und inhaltliche Überarbeitung** des Aufgaben-Fundus für den Übungsbetrieb zur Vorlesung Informatik 3 einschließlich des **Entwurfs neuer Übungsaufgaben** und der Entwicklung und Umsetzung einheitlicher Konzepte zur **Versionierung, Archivierung und dynamischen Vorschau**. Damit knüpft dieses Dokument an eine Arbeit von Tim Furche im März 2001 mit dem Titel „Entwicklung von Übungsaufgaben für die Vorlesungen TGI und Informatik 3“ an.

**Zum Vorgehen:** Zunächst wird der Ist-Zustand des Repositorys analysiert und bewertet. Dies geschieht unter Berücksichtigung der Erfahrungen der letzten Jahre und der durch die Arbeitsprozesse gegebenen Anforderungen. Aufbauend darauf werden die Zielsetzungen für die Neugestaltung formuliert und in ein Konzept umgesetzt. Darauf folgt der Entwurf neuer Aufgaben zu einzelnen Themenbereichen. Die Arbeit schließt ab mit einer Java-basierten Implementierung zum computergestützten Aufgaben-Entwurf.

## 2 Bisheriger Zustand des Aufgaben-Fundus

### 2.1 Aufgaben

Generell werden vier Typen von Aufgaben unterschieden:

- **Hausaufgaben (H):** Diese Aufgaben sind von geringem bis mittlerem Schwierigkeitsgrad und haben zumeist wiederholenden Charakter. Im Normallfall besteht eine Abgabepflicht für Aufgaben dieses Typs.

- **Klausuraufgaben (K):** Sie dienen den Studenten zur Orientierung sowie zur Prüfung ihrer Kenntnisse. Der Schwierigkeitsgrad dieser Aufgaben ist auf Klausurniveau.

- **Tutoraufgaben (T):** Im Wesentlichen neue, zum Teil auch in der Vorlesung nicht besprochene Lerninhalte und komplexere Problemstellungen werden in den so gekennzeichneten Aufgaben behandelt. In der Übungsgruppe erfolgt eine ausführliche Besprechung, der Schwierigkeitsgrad variiert von mittel bis sehr hoch.
- **Programmieraufgaben (P):** Sie dienen zur praktischen Umsetzung der in Vorlesung und Übung erlernten Inhalte. Als Programmiersprache für die Vorlesung Informatik 3 wurde Java gewählt, da sie alle entscheidenden Techniken und Konzepte bereit hält, die für einen Lernerfolg zielführend sind (entscheidend dabei: das Thread-Konzept und Synchronisation über Monitore).

Im Juli 2004 besteht der Aufgabenfundus für Informatik 3 aus etwa 100 Übungsaufgaben, die ein breites Spektrum des vorgesehenen Lernstoffs der Vorlesung abbilden. Allerdings sind große Teile der Aufgaben schon einige Jahre alt, was besonders bei Themen wie Betriebssystem-Grundlagen oder Dateisystemen schnell dazu führt, dass veraltete Konzepte oder Technologien behandelt werden.

### 2.2 Verwaltung der Quelldateien

In einem Semester werden zwischen 12 und 14 Übungsblätter ausgegeben. Die bisherige Verzeichnisstruktur basiert auf der Idee, für jedes dieser Blätter einen eigenen Ordner anzulegen, in dem sich die LaTeX-Quellen, Bilder/Grafiken und Java-Programme befinden, die für dieses Übungsblatt zur Verfügung stehen. In jedem Semester wird aus diesen Quellen eine Auswahl getroffen, die die tatsächliche Zusammenstellung des jeweiligen Übungsblattes darstellt. In einem Semester neu erstellte Aufgaben werden in den entsprechenden Ordnern abgelegt. Zu Beginn eines Semesters wird der komplette Fundus des vergangenen Semesters kopiert und ggf. um neue Aufgaben ergänzt. Häufig werden vorhandene Aufgaben auch variiert, aktualisiert oder korrigiert.

Bereits nach wenigen Semestern muss man feststellen, dass diese Form der Aufgaben-Verwaltung beträchtliche Probleme verursacht. Ändert sich die Reihenfolge der Lernstoffpräsentation in der Vorlesung, so müssen auch die Übungsblätter entsprechend angepasst werden. Das führt dazu, dass zum Beispiel für die Erstellung von Blatt 3 Aufgaben aus dem Ordner für Blatt 9 entnommen werden. Dem wiederum geht aufgrund der schlechten Dokumentation ein zum Teil erheblicher Suchaufwand voraus. Neue Aufgaben landen in falschen Ordern und bestehende Aufgaben sind plötzlich in mehr als einem Ordner in mehrfacher Ausfertigung zu finden. Es wird mit der Zeit immer schwerer, Duplikate von Variationen zu unterscheiden. Ein fehlendes, einheitlich umgesetztes Konzept zur Versionierung stiftet zusätzliches Chaos.

Im Juli 2004 besteht das gesamte Repository aus einer Ansammlung von Verzeichnissen und Quelldateien, in der fast keine Ordnung oder Struktur mehr erkennbar ist.

## 3 Strukturelle Überarbeitung: Zielsetzung

### 3.1 Klares, durchgängiges Verzeichnisdesign

Die Anforderungen an den Aufgabenfundus sind im Wesentlichen:

- einfaches Finden passender Übungsaufgaben aus dem bestehenden Fundus zu bestimmten Vorlesungsabschnitten,

- schnelle Identifikation von verschiedenen Versionen oder Varianten von Aufgaben und einfache Auswahl der zugehörigen Quelle,
- einfache und schnelle Aufnahme neu erstellter Übungsaufgaben in den Fundus.

Eine intuitive Verzeichnisstruktur und eine simple, aber effektive Versionierung sind die wichtigsten Schritte bei der Neugestaltung. Das bisherige Konzept wurde zu diesem Zweck vollständig verworfen, alle Aufgaben gesichtet und eine neue Struktur aufgebaut, die sich nicht am zeitlichen Verlauf des Semesters, sondern an den Themen der Vorlesung orientiert.

## 3.2 Themengebiete

Folgende Themengebiete wurden identifiziert:

1. Betriebssysteme
2. Unix-Grundlagen
3. Prozesse
4. Threads
5. Scheduling
6. Scheduling-Simulation
7. Deadlocks
8. Prozesskoordination
9. Speicher
10. Dateisysteme
11. Verteilte Systeme
12. Java Vermischt
13. Wiederholungs-Aufgaben (decken mehrere Themenbereiche ab)

## 3.3 Neue Verzeichnisstruktur

Die neue Verzeichnisstruktur entspricht genau diesen Themen. So befinden sich zum Beispiel alle Aufgaben zum Thema Prozess-Scheduling im Verzeichnis mit dem Namen „Scheduling“ usw. Jeder dieser Themenordner enthält also

- die LaTeX-Quell-Dateien aller Aufgaben zu diesem Themenbereich,
- ein Unterverzeichnis „Pics“, das sämtliche Bilder und Grafiken enthält, die in den Aufgaben oder Musterlösungen der Aufgaben dieses Themenbereichs eingebunden werden, sowie
- ein Unterverzeichnis „Sources“, das die den Programmieraufgaben zugehörigen Java-Klassen und Programmrahmen enthält.

Alle Themenordner befinden sich im Ordner „.repos“ (für „Repository“). „.repos“ selbst ist Unterverzeichnis im Ordner „aufgaben“ und teilt sich diesen mit verschiedenen Makro-Ordnern (in denen im Wesentlichen Layout-Einstellungen in Form von Stylesheets gespeichert werden, die für die Arbeit des Betreuuers der Vorlesung praktisch keine Rolle spielen), dem Preview-Ordner (siehe Kapitel 4) und Verzeichnissen für jedes Semester, in dem ein Übungsbetrieb stattfindet bzw. stattfand. Es liegt auf der Hand, dass diese Semester-Ordner nun die ganz konkreten Aufgabenzusammenstellungen für die einzelnen Übungsblätter in dem jeweiligen Semester enthalten. Jeder Semester-Ordner (zum Beispiel „ws0405“) enthält für jedes Übungsblatt (zum Beispiel „blatt01“ bis „blatt14) ein eigenes Unterverzeichnis. Und in diesen Blatt-Ordnern wird nun alles gespeichert, was zur Erstellung eines Übungsblatt benötigt wird (wie das Makefile, das Body-File usw.).

Wichtig: Außerhalb des Ordners „.repos“ befinden sich keine Quell-Dateien von Aufgaben oder Lösungen. Auf den nächsten zwei Seiten ist die vollständige Datei- und Verzeichnishierarchie noch einmal graphisch dargestellt.

```
Info3
  |
  |--- aufgaben
  |       |
  |       |--- .klausurmakros
  |       |       |
  |       |       |--- <Style-Templates für die Klausur>
  |       |       |--- ...
  |       |
  |       |--- .makros
  |       |       |
  |       |       |--- <Allgemeine Style-Templates>
  |       |       |--- ...
  |       |
  |       |--- .repos
  |       |       |
  |       |       |--- 01_Betriebssysteme
  |       |       |       |
  |       |       |       |--- Pics
  |       |       |       |       |
  |       |       |       |       |--- BSSchichten.fig
  |       |       |       |       |--- BSSchichten.eps
  |       |       |       |       |--- ...
  |       |       |       |
  |       |       |       |--- Sources
  |       |       |       |       |
  |       |       |       |       |--- test.java
  |       |       |       |       |--- ...
  |       |       |       |
  |       |       |       |--- H_BSEbenen.tex
  |       |       |       |--- H_BSGrundlagen.tex
  |       |       |       |--- K_MC_Powermanagement_1.tex
  |       |       |       |--- ...
  |       |       |
  |       |       |--- 02_UnixGrundlagen
  |       |       |       |
  |       |       |       |--- <analog zu 01_Betriebssysteme>
  |       |       |
  |       |       |--- 03_Prozesse
  |       |       |--- 04_Threads
  |       |       |--- 05_Scheduling
  |       |       |--- 06_SchedulingSimulation
  |       |       |--- 07_Deadlocks
  |       |       |--- 08_Prozesskoordination
  |       |       |--- 09_Speicher
  |       |       |--- 10_Dateisysteme
  |       |       |--- 11_VerteilteSysteme
  |       |       |--- 12_JavaVermischt
  |       |       |--- 13_Wiederholung
  |       |
```

```
|       |--- preview
|       |       |
|       |       |--- <Preview-Dateien für Aufgaben-Vorschau>
|       |       |--- ...
|       |
|       |--- ws0405
|       |       |
|       |       |--- blatt01
|       |       |       |
|       |       |       |--- .tmp
|       |       |       |       |
|       |       |       |       |--- <Temporäre Dateien>
|       |       |       |
|       |       |       |--- Makefile
|       |       |       |--- blatt01.answer.ps
|       |       |       |--- blatt01.answer.tex
|       |       |       |--- blatt01.ps
|       |       |       |--- blatt01.tex
|       |       |       |--- body.tex
|       |       |
|       |       |--- blatt02
|       |       |       |
|       |       |       |--- <analog zu blatt01>
|       |       |
|       |       |--- ...
|       |       |
|       |       |--- blatt14
|       |               |
|       |               |--- <analog zu blatt01>
|       |
|       |--- ws0506
|       |       |
|       |       |--- <analog zu ws0405>
|       |
|       |--- ...
|
|--- Progs
        |
        |--- info3.cgi
        |--- tgi.cgi
        |--- ...
```

## 3.4 Dateinamen-Design & Versionierung

Die Dateinamen für die Quell-Dateien der Aufgaben werden einheitlich formatiert. Dabei wird folgendes Muster verwendet:

`<Typ>_<Zusatz>_<Titel>_<Variante>_<Version>.tex`

- **Typ:** H (Haus-), K (Klausur-), P (Programmier-) oder T (Tutoraufgabe)

- **Zusatz:** MC (Multiple Choice), WDH (Wiederholung) oder nichts
- **Titel:** Ein aussagekräftiger Titel für die Übungsaufgabe
- **Variante:** Varianten von Übungsaufgaben werden beginnend mit 1 durchnummeriert.
- **Version:** Versionen von Übungsaufgaben werden beginnend mit V1 durchnummeriert.

**Variante vs. Version:** Diese Differenzierung macht Sinn, um folgendes zu gewährleisten: Bei verschiedenen Versionen einer Aufgabe soll davon ausgegangen werden, dass immer die letzte Version auch die aktuellste ist und damit allen Vorgängerversionen vorzuziehen ist. Verschiedene Varianten hingegen können sich in ihrer Aufgabenstellung unterscheiden und prinzipiell komplett verschiedene Aufgaben sein (für die lediglich keine verschiedenen Titel gefunden werden konnten).

**Beispiele:**

- Von einer Multiple-Choice-Klausuraufgabe zum Thema „Deadlocks" gibt es zwei Versionen, da ein Fehler aus der ersten Version korrigiert wurde:

  ```
  K_MC_Deadlocks_V1.tex
  K_MC_Deadlocks_V2.tex
  ```

- Es gibt drei Programmieraufgaben zum Thema „Java-Monitore", wobei eine dieser Varianten in wiederum zwei Versionen vorliegt:

  ```
  P_JavaMonitore_1.tex
  P_JavaMonitore_2.tex
  P_JavaMonitore_3_V1.tex
  P_JavaMonitore_3_V2.tex
  ```

**Beispiel: Inhalt des Themen-Ordners „Scheduling"**

```
H_Dispatcher.tex
H_MC_Scheduling_1.tex
H_MC_Scheduling_2.tex
H_MC_Scheduling_3.tex
H_RoundRobin_V1.tex
H_RoundRobin_V2.tex
H_WDH_SchedulingAllgemein.tex
K_SchedulingStrategienVergleich.tex
T_GrahamsList_V1.tex
T_GrahamsList_V2.tex
```

# 4 Dynamische Übersichts- und Vorschaufunktionen

## 4.1 Ausgangssituation und Zielsetzung

Die Themen-orientierte Verzeichnisstruktur macht es zwar einfach, Aufgaben zu einem bestimmten Thema zu finden, allerdings muss dazu händisch in den Ordnern des Repositorys navigiert und gesucht werden. Eine grobe Selektion erfolgt über die Dateinamen. Um sich den Inhalt einer Aufgabe (und ggf. deren Lösung) anzusehen, könnte man die Quelldatei mit einem Text-Editor öffnen. Das Lesen im Quelltext kann aber mühsam und zeitraubend sein. Besser wäre eine fertig kompilierte Vorschau im Postscript- oder PDF-Format, die die Übungsaufgabe so darstellt, wie sie auch auf einem Übungsblatt aussehen würde. Für diese Art der Vorschau müsste man ein Dummy-Aufgabenblatt erstellen, was mit einigen Minuten Aufwand verbunden ist. Automatisierte Vorschau-Funktionen machen das künftig unnötig. Diese wiederum sind Teil einer dynamischen Übersicht im HTML-Format.

**Hinweis zur Vorlesung „Technische Grundlagen der Informatik" (TGI):** Die jetzt beschriebenen Funktionen lassen sich in gleicher Weise auf den Aufgaben-Fundus der TGI-Vorlesung anwenden, das in einer ebenfalls themenorientierten Verzeichnisstruktur vorliegt.

## 4.2 Installation und Einrichtung

Basiskonfiguration:

- Das Gesamt-Repository wird ab jetzt von einem Linux-Webserver (zum Beispiel Apache) verwaltet. Dazu wird der komplette Ordner „Info3" (bzw. „TGI") in das Arbeitsverzeichnis der Webservers verschoben (bzw. der Webserver entsprechend eingestellt).
- Für das Verzeichnis `Info3/preview/` (bzw. `TGI/preview/`) sollten alle Zugriffsbeschränkungen aufgehoben werden (volle Lese- und Schreibrechte für alle Nutzer und Gruppen). In allen anderen Verzeichnissen sollten die Zugriffs- (darunter natürlich im Besonderen die Schreib-) Rechte aus Sicherheitsgründen so restriktiv wie möglich gehalten werden.
- Sämtliche Dateien aus dem Ordner `Progs/` müssen ins Verzeichnis `cgi-bin/` des Webservers kopiert werden. Im Einzelnen:
  - `info3.cgi` und `tgi.cgi`
  - `makeps.cgi` und `makeps2.cgi`
  - `printrepos.cgi` und `printrepos2.cgi`
- Natürlich muss der Webserver noch gestartet werden.

Mehr ist nicht zu tun. Der folgende Aufruf zeigt die nach Themen sortierte Aufgabenübersicht für die Vorlesung „Informatik 3" im Web-Browser an:

`http://localhost/cgi-bin/info3.cgi`

Dieser Aufruf gilt für den Fall, dass Server und Client auf dem gleichen Rechner liegen. Liegt der Webserver auf einem anderen als dem aufrufendem Rechner, so muss `localhost` durch den Namen dieses Rechners ersetzt werden. Für das TGI-Repository gilt analog der Aufruf:

`http://localhost/cgi-bin/tgi.cgi`

## 4.3 Funktionen

Im Web-Browser werden nach den genannten Aufrufen die Titel (Dateinamen) aller Übungsaufgaben in alphabetischer Reihenfolge angezeigt. An dieser Stelle gibt es drei Optionen:

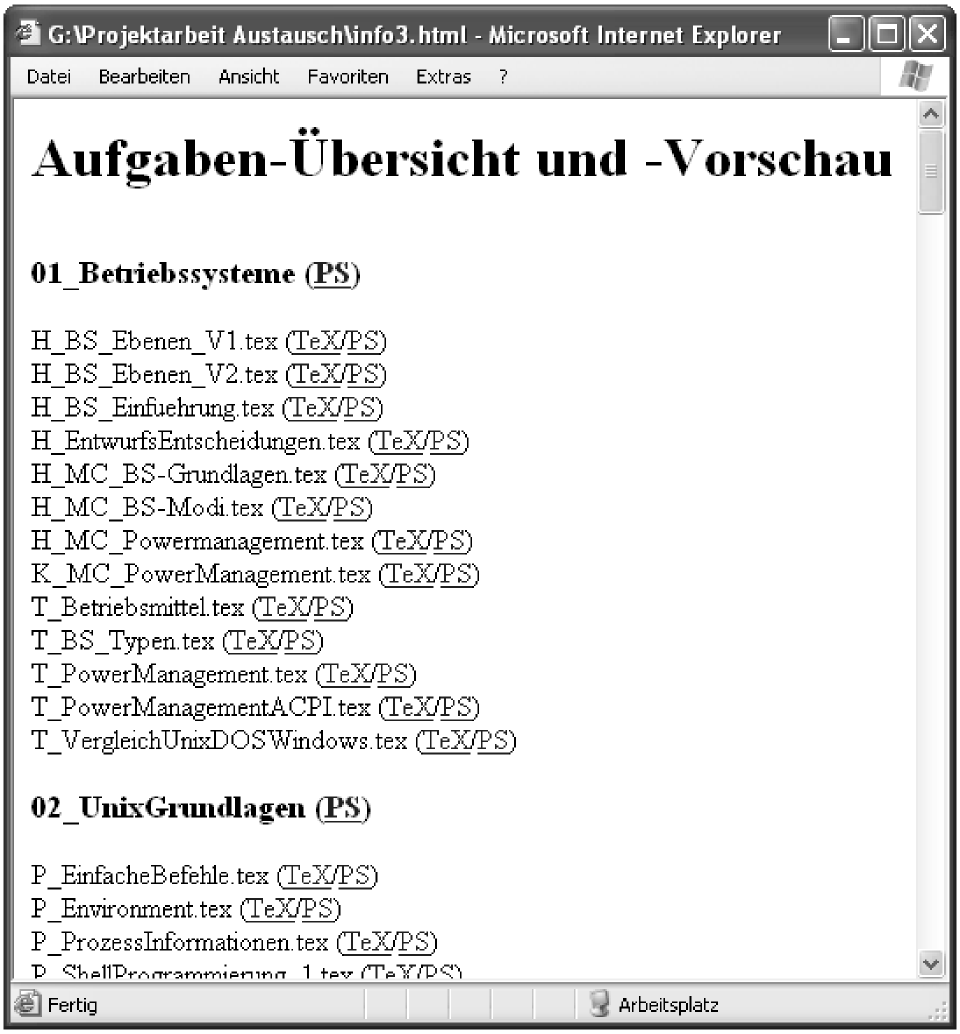


Abbildung 1: Aufgaben-Übersicht im HTML-Browser

1. **Den LaTeX-Quellcode zu einer Aufgabe anzeigen:** Ein Klick auf den `TeX`-Link neben einem Dateinamen öffnet den unkompilierten Quellcode zu einer Übungsaufgabe. Diese Option kann interessant sein, um den Metadaten-Header einer Aufgabe zu betrachten oder um die Aufgabe als Vorlage zu verwenden. Für eine kompilierte Vorschau sollte hingegen die nächste Option verwendet werden.

2. **Postscript-Vorschau zu einer Aufgabe anzeigen:** Nach Klick auf den `PS`-Link neben einem Dateinamen werden automatisch zwei Postscript-Dokumente zu der gewählten Übungsaufgabe angelegt. Parallel öffnet sich eine neue HTML-Seite im Browser. Sie bietet die Wahl zwischen einer Ansicht ohne bzw. mit Musterlösung.

3. **Alle Aufgaben eines Themenbereichs anzeigen:** Zu dieser Vorschaufunktion führt der `PS`-Link neben einem Thema. Anschließend besteht wieder eine Auswahlmöglichkeit zwischen einer Ansicht mit oder ohne Lösungen.

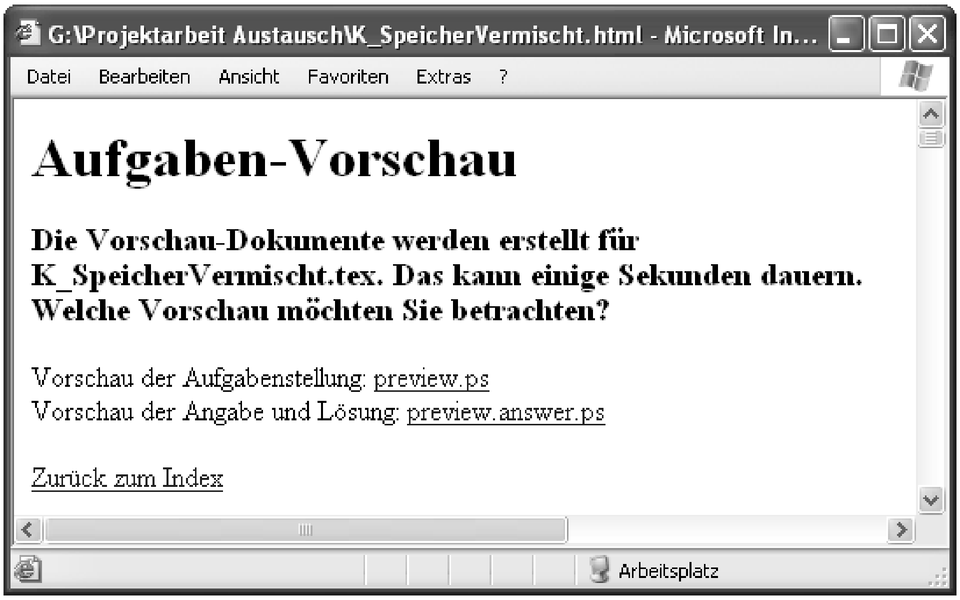


Abbildung 2: Auswahlmöglichkeiten bei dynamischer Vorschau

### 4.4 Implementierung

Für die Lösung dieser Aufgabe wurden zwei Sprachen eingesetzt: HTML zur Darstellung der Nutzerschnittstelle (als Webseite) und Perl für die Verarbeitung der dynamischen Information. Die Umsetzung erfolg über das CGI (Common Gateway Interface) eines (prinzipiell beliebigen) Webservers. Die Anfrage `http://localhost/cgi-bin/info3.cgi` bewirkt die serverseitige Ausführung eines Perl-Skripts, welches die aktuelle Ausprägung des Repositorys einliest und aus diesen Informationen die HTML-Ansicht aller Themen und Aufgaben erstellt:

```
#!/usr/bin/perl -w

use strict;
use CGI::Carp qw(fatalsToBrowser);

print "Content-type: text/html\n\n";
print '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">', "\n";
print "<html><body><h1>Aufgaben-Übersicht und -Vorschau</h1><br/>\n\n";

chdir "./../htdocs/Info3/aufgaben/.repos" or die "Fehler beim Wechseln des
Verzeichnisses: $!";
my $count = 0;

my @topics = <*>;
foreach (@topics) {
  next if $_ eq "_VorlesungsParameter.tex";
  next if $_ eq "RepositoryPDF";
  print "<h3>$_ (<a href=\"http://localhost/cgi-bin/printrepos.cgi/$_\">PS</a>)</h3>\n";
  my $father = $_;
```

```
  chdir "./$_";
  my @sources = <*>;
  foreach (@sources) {
    next if $_ eq "Pics";
    next if $_ eq "Sources";
    $count = $count + 1;
    print "$_ (<a href=\"http://localhost/Info3/aufgaben/.repos/$father/$_\">TeX</a>
    /<a href=\"http://localhost/cgi-bin/makeps.cgi/$_\">PS</a>)<br/>\n";
  }
  print "<br/>\n\n";
  chdir "./..";
}

print "$count Aufgaben insgesamt.<br/>";
print "</body></html>";
```

# 5 Entwurf neuer Übungsaufgaben

## 5.1 Überblick

Im Rahmen dieser Projektarbeit wurden 66 neue Übungsaufgaben zur Vorlesung „Informatik 3“ erstellt – davon

- 11 Aufgaben als Erweiterungen zu bereits bestehenden Übungen.
- 55 Aufgaben ohne vergleichbare Vorlage im bisherigen Fundus und
- insgesamt 23 Aufgaben im Multiple-Choice-Stil (entspricht etwa 115 Einzel-Statements).

Dieses Kapitel der Ausarbeitung gibt einen groben und somit keineswegs vollständigen Überblick über die neu erstellten Übungsaufgaben. Der Fokus hierbei liegt auf der Darstellung von Methodik, Didaktik und Zielen am Beispiel konkreter Aufgabenstellungen einzelner Themenbereiche. Sämtliche neuen Aufgaben wurden in gedruckter und digitaler Form gemeinsam mit dieser Ausarbeitung eingereicht.

## 5.2 Einführende Aufgaben

### 5.2.1 Zielsetzung der einführenden Aufgaben

Die einführenden Aufgaben, die in der Regel vor allem die ersten beiden Übungsblätter eines Semesters prägen, sollten einen Einblick darüber geben, was die Vorlesung Informatik 3 inhaltlich vermitteln will und worin Bedarf und Nutzen für diese Inhalte bestehen. Diese Problemorientierung fehlt einem Großteil der bisherigen Aufgaben allerdings, und sei es nur deshalb, weil manche diese Aufgaben veraltet sind und mit heutigen Betriebssystemen nicht mehr viel zu tun haben. Darüberhinaus erscheinen einige Aufgaben zu „konzeptionell“ und abstrakt, während andere wiederum zu stark ins Detail gehen und auf eigentlich unwesentliche Lerninhalte fokussiert sind. Manche Aufgaben aus dem Unix-Umfeld sind schlecht motiviert oder überspringen wichtige Grundlagen, die einigen Studenten im dritten Semester aufgrund der hohen Verbreitung von Windows-Rechnern schlicht fehlen.

**Themen bisheriger Einführungs-Aufgaben:**

- Betriebssystem-Komponenten
- Betriebssystem-Schichten
- Vergleich von MS DOS, Windows 95 und Linux

### 5.2.2 Übung 1: Hauptaufgaben von Betriebssystemen

Diese Aufgabe wurde als umfassende Einführung in das Thema „Betriebssysteme“ entwickelt. Sie greift auch einzelne Module aus den bestehenden Basis-Aufgaben auf und fasst sie in integrierter Weise zu einer durchgängigen Übung zusammen. Die wesentlichen, großen Themenbereiche der Vorlesung, wie zum Beispiel Multiprogramming, Ressourcen- und Speicherverwaltung oder Dateisysteme, werden angesprochen.

**Aufgabe:**

1. Welche Charakteristika, die bei der Arbeit mit einem Desktop-PC heute selbstverständlich sind, werden erst durch das Betriebssystem ermöglicht?
2. Leiten Sie daraus die Haupt-Aufgaben eines modernen Betriebssystems ab.
3. Erklären Sie kurz die folgenden Begriffe aus der Betriebssystem-Welt:
   (a) Multi-Tasking
   (b) Scheduling und Dispatching
   (c) Speicherverwaltung
   (d) Virtueller Speicher
   (e) Dateisystem
   (f) Shell
4. Vergleichen Sie anhand selbst gewählter Kriterien grob die Betriebssysteme MS DOS, Windows XP und Linux.

### 5.2.3 Übung 9: Grundlagen der Shell-Programmierung

Der gebräuchlichste Kommandozeilen-Interpreter unter Linux ist die `bash`-Shell. Während die Shell als Schnittstelle zwischen Nutzer und Betriebssystem bzw. Rechner bereits in den konzeptionellen Aufgaben behandelt wurde, sollen nun Grundlagen der Shell-Programmierung vermittelt werden, indem einfache Skripte programmiert werden, die verschiedene Problemstellungen lösen.

Folgende Konzepte der Shell-Programmierung werden schrittweise vermittelt:

- Aufbau und Ausführung von Skripten
- Shellbefehle
- Variablen und Ausgabe-Routinen
- Pipes, Redirection und Backtick
- Steuerungsstrukturen (Case, If-then-else, Schleifen)
- Quotierung
- Funktionen
- Kommandozeilenargumente

**Beispiel:** Skript, das den Typ einer Datei (ZIP oder Text) automatisch erkennt und die Datei in ein geeignetes Programm lädt

```
#!/bin/sh
ftype=`file "$1"`
case "$ftype" in
"$1: Zip archive"*)
    unzip "$1" ;;
"$1: ISO-8859 text"*)
    xemacs "$1" ;;
*) error "Datei $1 ist weder ZIP- noch Textdatei";;
esac
```

## 5.3 Programmierprojekt: „Scheduler-Simulation“

### 5.3.1 Motivation

Die Programmieraufgaben geben den Studenten Gelegenheit, einige der Konzepte des Betriebssystem-Designs in die Praxis umzusetzen. Manche dieser Aufgaben (insbesondere zu Threads und Synchronisation) erscheinen dabei zwar recht anschaulich, allerdings weit weg vom Thema „Betriebssysteme“, dem eigentlichen Fokus der Informatik 3-Vorlesung.

Die im Folgenden entwickelte Übungsaufgabe – bestehend aus mehreren Teilaufgaben – hat das Ziel, einige der wichtigsten Themen der Vorlesung, nämlich das Prozess-Konzept, die Prozess-Zustands-Modelle sowie ganz besonders das Prozess-Scheduling, mit den vermittelten Java-Programmierkenntnissen zu verknüpfen. Im Unterschied zu fast allen existenten Programmieraufgaben wurde sie so gestaltet, dass die einzelnen Teilschritte auf dem Weg zur Gesamtlösung zum einen Teil als Tutoraufgaben und zum anderen Teil als Hausaufgaben gestellt werden können. Die Herausforderung liegt sicherlich auch darin, eine Motivation zu schaffen, die dazu beiträgt, dass sich möglichst viele Studenten selbständig mit der Lösungssuche und Programmierversuchen beschäftigen. Ein Problem der Programmieraufgaben ist erfahrungsgemäß, dass sie eher als „lästiges Beiwerk“ denn als nutzbringende Lernerfahrung wahrgenommen werden. Man könnte vielleicht sagen: Dies ist ein Versuch, das „Image-Problem“ der Programmieraufgaben abzumildern.

### 5.3.2 Übung 21: Vorüberlegungen

In einer ersten Teilaufgabe, die wahlweise als Tutor- oder als Hausaufgabe definiert werden kann, sollen die Grundlagen für die Simulation eines (immerhin bereits preemptiven) Priority-Schedulers erarbeitet werden. In dieser Aufgabe wird noch nicht programmiert.

> **Aufgabe:** Sie haben verschiedene Arten von Schedulern (preemptiv, nicht-preemptiv) sowie unterschiedliche Scheduling-Strategien (Prioritäten, FCFS, Round Robin usw.) in der Theorie kennen gelernt. Nun sollen Sie selbst einen Scheduler in der Programmiersprache Java implementieren. Es ist klar, dass es sich dabei nur um eine Simulation, nicht um einen realen und funktionsfähigen Scheduler handeln kann. In dieser Aufgabe sollen Sie noch nicht programmieren, sondern einige Vorüberlegungen anstellen, die Sie dann in weiteren Teilaufgaben schrittweise umsetzen werden. (...)

In dieser Aufgabe werden Überlegungen dazu angestellt,

- inwiefern Java überhaupt dazu geeignet ist, einen Scheduler zu simulieren,
- wie Unterbrechungen nach Zeitintervallen in realen Systemen realisiert und in Java simuliert werden können,

- wie die laufenden Prozesse in objektorientierter Weise realisiert werden können,
- welche Datenstruktur sich für die Prozessverwaltung eignet und
- wie sich mit Hilfe des Vererbungskonzepts unterschiedliche Scheduling-Strategien ohne Programmcode-Dopplungen implementieren lassen.

### 5.3.3 Übung 22: Prozessdeskriptor

Aufbauend auf den Vorüberlegungen aus der vorangegangenen Teilaufgabe soll in dieser kurzen Aufgabe eine Klasse für den allgemeinem, Strategie-unabhängigen Prozessdeskriptor geschrieben werden. Dabei müssen Überlegungen dazu angestellt werden, welche Prozessattribute bei jeder Scheduling-Strategie benötigt werden, und welche erst später in die den Strategien angepassten Unterklassen eingebettet werden. Zur Erfassung von Ankunfts- und Abgangszeitpunkten einzelner Prozesse, die später die Grundlage für die Berechnung von Verweil- und Wartezeiten bilden, werden die Studenten in dieser Aufgabe mit der *java.util.Date*-Klasse vertraut gemacht.

### 5.3.4 Übung 23: Priority-Scheduler

An diesem Punkt sind die Variationsmöglichkeiten vielfältig. Um die Aufgabe des Entwurfs einer lauffähigen Scheduler-Simulation auf der Grundlage der bisherigen Ergebnisse als Hausaufgabe zu geben, wird man am besten größere Teile des Codes der fehlenden Klassen bereits vorgeben und stellenweise zu ergänzende Programmfragmente markieren.

```
public class PriorityScheduler extends Scheduler
{
  private PriorityProcess current;

  public PriorityScheduler() {
    super(); //Aufruf: Konstruktor der Oberklasse
    System.out.println("Scheduling-Strategy: Priority-Scheduling");
  }

  public void run() { //Scheduler läuft als ständiger Thread
    while(true) { //in einer Endlosschleife
      try {
        if(current == null) { //kein Prozess aktiv
          current = PriorityProcess.getFirst();
          //zu aktivierender Prozess = erster in Liste
          System.out.print("Scheduler decided: Start with first process in queue: ");
          current.printInfo(); //Prozess-Informationen ausgeben
          current.start(); //Prozess starten
        }
        else { //bereits ein Prozess aktiv
          if(!current.isAlive()) {
  //aktiver Prozess lebt nicht mehr, ist also fertig
            current.removeProcess(); //Prozess aus Liste entfernen
            current = PriorityProcess.getFirst();
    //zu aktivierender Prozess = nächster in Liste

            if(current.isSuspended()) {
```

```
    //zu aktivierender Prozess wurde zuvor suspendiert
              System.out.print("Scheduler decided: Restore process: ");
              current.printInfo(); //Prozess-Informationen ausgeben
      current.unsetSuspended(); //Suspend-Flag entfernen (suspended = false)
      current.resume(); //Prozess wiederherstellen
    }
    else {
    //zu aktivierender Prozess wurde nicht suspendiert (ist also "neu")
              System.out.print("Scheduler decided: Switch to next process in queue: ");
              current.printInfo(); //Prozess-Informationen ausgeben
      current.start(); //Prozess starten
    }
  }

          if(current != PriorityProcess.getFirst()) {
  //aktiver Prozess entspricht nicht erstem in Queue
            /////////////////////////////////
            //                             //
            //                             //
            // Zu ergänzender Programmteil //
            //                             //
            //                             //
            /////////////////////////////////
          }
          else {
  //aktiver Prozess entspricht auch erstem in Queue, also beibehalten
            System.out.print("Scheduler decided: Keep process ");
    current.printInfo(); //Prozess-Informationen ausgeben
          }
        }
        Thread.sleep(20);
      }
      catch (Exception e) {
        System.out.println("Nothing more to do.");
        System.exit(0); //Simulation beenden (alle Prozesse terminiert)
      }
    }
  }
}
```

#### 5.3.5 Übung 24: „First Come First Serve-Scheduler“

Mit dem Priority-Scheduler wurde ein preemptiver Scheduling-Algorithmus realisiert. Im Kontrast dazu steht die Klasse nicht-unterbrechender Scheduling-Routinen, zu der die Strategie „First Come First Serve“ gehört. Wegen des geringeren Schwierigkeitsgrades eignet sich diese Teilaufgabe für eine Klausur.

## 5.4 Paging-Systeme

### 5.4.1 Motivation

Das Paging-System ist ein wichtiges Modul jedes Betriebssystems. Punktuell wird dieses Thema durch bestehende Übungsaufgaben erfasst, im Wesentlichen mit Blick auf die Seitenersetzungsstrategien (ähnlich wie im Kapitel Prozess-Scheduling die Scheduling-Strategien). Die klassische Aufgabe zum Thema Paging sieht wie folgt aus: Eine Reihe von Speicherzugriffen sowie die Größe des verfügbaren Speichers (als Anzahl der Seitenrahmen) sind gegeben. In tabellarischer Form soll ermittelt werden, wie viele Seitenfehler mit den verschiedenen Ersetzungsalgorithmen (First In First Out, Last In First Out, Least Recently Used, Least Frequently Used) erzeugt werden. Auf der Basis dieser Beobachtungen sollen die unterschiedlichen Algorithmen bewertet werden. Die Faustregel: Je weniger Seitenfehler, umso besser.

Aufgaben in diesem Stil vermitteln das grundlegende Konzept des virtuellen Speichers und der Auslagerung von Seiten. Sie forcieren zudem eine Auseinandersetzung mit den gängigsten Paging-Strategien. In [1, Moderne Betriebssysteme] wird an diesem Punkt tiefer in die Theorie der Speicherverwaltung eingedrungen. Zunächst werden die Seitenersetzungsalgorithmen in zwei Klassen eingeteilt: die Klasse der Keller-Algorithmen und die Klasse aller restlichen Algorithmen. Diese Differenzierung wird anhand eines Beispiels motiviert. Außerdem wird mit der Distanzkette eine abstrakte Repräsentation der Referenzkette eingeführt. Der Vorteil dieses Vorgehens ist, dass ein Paging-System nicht mehr wie bisher durch die Liste der Speicherzugriffe (Referenzkette) und der zu Grunde liegenden Seitenersetzungsstrategie charakterisiert wird, sondern nur noch durch die Distanzkette, die sowohl von der Referenzkette, als auch von der Strategie abhängt. Auf dieser Grundlage ist es möglich, jedes Paging-System in einheitlicher Weise zu bewerten. Die Seitenfehlerraten können für beliebige Speichergrößen vektoriell ermittelt (vorhergesagt) werden.

Zwei neue Aufgaben zum Thema Seitenersetzung beschäftigen sich mit diesen Konzepten.

### 5.4.2 Übung 50: Abstrakter Interpreter

Bei der ersten Aufgabe handelt es sich um eine Tutor-Aufgabe, Vorwissen aus der Vorlesung ist (abgesehen von den Grundlagen) nicht erforderlich. Die Aufgabe ist folgendermaßen aufgebaut:

- Zunächst wird ein abstrakter Interpreter für die Seitenersetzungsstrategie LRU (Least Recently Used) entwickelt, der auf einer internen Tabelle basiert, die Haupt- und Hintergrundspeicher zusammenfassend repräsentiert.
- Dieser Interpreter muss nun eine gegebene Referenzkette (also Folge von Speicherzugriffen) abarbeiten. Dabei wird die Anzahl der auftretenden Seitenfehler festgehalten.
- Durch dieses Beispiel motiviert wird die Klasse der Keller-Algorithmen zunächst informell, dann formal-genau spezifiziert.
- Nun wird begründet, warum bei Keller-Algorithmen keine Seitenfehler-Anomalie (Beladys-Anomalie) eintreten kann.
- Als nächstes wird das Konzept der Distanzkette eingeführt, definiert und die Distanzkette für das Beispiel in dieser Aufgabe ermittelt.

- Und schließlich wird der Algorithmus entwickelt, der es auf der Grundlage einer Distanzkette möglich macht, die Anzahl der Seitenfehler für verschiedene Speichergrößen vorherzusagen.

### 5.4.3 Übung 51: Keller-Algorithmen und Beladys-Anomalie

Die zweite Aufgabe basiert auf den Ergebnissen des abstrakten Interpreters. Sie ist auf Hausaufgaben-Niveau gestellt und behandelt zwei Aspekte:

- Wiederholung des Konzepts der Distanzkette und
- Einordnung der FIFO-Strategie in die bekannten Klassen von Seitenersetzungsalgorithmen (mit Beweis).

> **Aufgabe:** Beweisen Sie, dass der FIFO-Algorithmus zur Seitenersetzung kein Keller-Algorithmus ist.
>
> **Lösungsvorschlag:** Der „Beweis“ ist ein einfaches Beispiel, das zeigt, dass FIFO die Keller-Eigenschaft nicht erfüllt. Dazu überlegt man sich zunächst eine geeignete Referenzkette, wie:
>
> `w = 0 1 2 0 3 0 1 2 3`
>
> Nun ermittelt man tabellarisch, wie diese Referenzkette nach der FIFO-Strategie bei zwei bzw. drei Seitenrahmen abgearbeitet wird:

| 0 | 1 | 2 | 0 | 3 | 0 | 1 | 2 | 3 |
|---|---|---|---|---|---|---|---|---|
| 0 | 1<br>0 | 2<br>1 | 0<br>2 | 3<br>0 | 3<br>0 | 1<br>3 | 2<br>1 | 3<br>2 |
| | | 0 | 1 | 2<br>1 | 2<br>1 | 0<br>2 | 3<br>0 | 1<br>0 |
| P | P | P | P | P | | P | P | P |

| 0 | 1 | 2 | 0 | 3 | 0 | 1 | 2 | 3 |
|---|---|---|---|---|---|---|---|---|
| 0 | 1<br>0 | 2<br>1<br>0 | 2<br>1<br>0 | 3<br>2<br>1 | 0<br>3<br>2 | 1<br>0<br>3 | 2<br>1<br>0 | 3<br>2<br>1 |
| | | | | 0 | 1 | 2 | 3 | 0 |
| P | P | P | | P | P | P | P | P |

> Nach dem fünften Zugriff ergibt sich folgendes Bild: Bei zwei Seitenrahmen enthält der Speicher die Seiten $(0, 3)$, bei drei Seitenrahmen die Seiten $(1, 2, 3)$. Die Keller-Eigenschaft $M(m, r) \subseteq M(m + 1, r)$ ist nicht erfüllt, da $(0, 3)$ keine Teilmenge von $(1, 2, 3)$ ist. Und deswegen kann FIFO kein Keller-Algorithmus sein.

## 5.5 Dateisysteme und Recovery

### 5.5.1 Motivation

Bisherige Übungsaufgaben beschäftigen sich mit den grundlegenden Konzepten typischer Unix-Dateisysteme, wie Blöcke, I-Nodes und Link-Unterstützung. Welche Abhängigkeiten bestehen zwischen Parametern wie Blockgröße, Adressbreite, Partitionsgröße oder maximaler Dateigröße? Was versteht man unter Datenkonsistenz? Wie sehen Konzepte zur Behebung von Fehlern nach Systemcrashs oder Stromausfällen aus? Diese und andere Fragen werden in einigen neuen Aufgaben zum Thema „Dateisysteme“ aufgeworfen (und hoffentlich auch beantwortet). Im Zusammenhang mit Journaling wird auch das Transaktionskonzept behandelt.

### 5.5.2 Übung 57: Fallanalyse Ext2/Ext3

Lange Jahre war Ext2 (Second Extended Filesystem) bei Linux-Nutzern das Dateisystem erster Wahl. Es implementiert die klassischen Konzepte traditioneller Unix-Dateisysteme, wie Blöcke und I-Nodes. Ext3 basiert auf der gleichen Architektur, bietet aber zusätzlich Journaling. In Übung 57 werden an diesen konkreten Beispielen eingeführt: Block-Konzept, Wahl der Blockgröße und ihr Einfluss auf den Fragmentierungsgrad, Blockgruppen und Superblock, Begriff der Konsistenz, Metadaten vs. Nutzdaten, Transaktionskonzept und Logging/Journaling. (Das I-Node-Prinzip wird in einer separaten Übung ausführlich verdeutlicht.)

### 5.5.3 Übung 58: Modellierung eines inkonsistenten Zustands

Was bedeutet es, wenn ein Dateisystem „inkonsistent“ ist? Und wie kann sich das negativ auswirken? Diese Übungsaufgabe modelliert ein ganz konkretes Szenario, bei dem während einer Operation im Dateisystem ein Stromausfall auftritt und dadurch ein wichtiges I-Node-Update verloren geht. Ein Auszug:

> **Aufgabe:** Stellen Sie sich folgende Situation vor: Die Datei `myfile.html` (Dateigröße: 1,5 KB, I-Node-Nummer: 522, Datenblöcke: 20.392 und 20.393) liegt im Verzeichnis `myfolder` (Datenblock: 12.440). Der Referenzzähler im I-Node mit der Nummer 522 steht auf 1, das heißt es gibt keine weiteren Hardlinks auf `myfile.html`. Nun sollen nacheinander die folgenden beiden Aktionen durchgeführt werden:
>
> I.) Ein Hardlink auf die Datei `myfile.html` soll im Verzeichnis `myhome` (Datenblock: 16.500) angelegt werden.
>
> II.) Anschließend soll die Datei `myfile.html` aus dem Ordner `myfolder` entfernt werden.
>
> Nehmen Sie für diese Aufgabe vereinfachend an, dass das **Anlegen eines Hardlinks** im Dateisystem allgemein in zwei Schritten erfolgt:
>
> 1. Zuerst wird der Datenblock des Verzeichnisses, in dem der Hardlink angelegt wird, neu geschrieben.
> 2. Danach wird der I-Node der Datei, auf die der Hardlink erstellt wurde, aktualisiert.
>
> (...)
>
> Während der Aktion I des Szenarios (Hardlink anlegen) falle die Stromversorgung aus, und zwar unmittelbar nachdem der Teilschritt 1 (Datenblock neu schreiben) beendet wurde. Der Rechner wird ohne Dateisystem-Scan neu gestartet. Was passiert, wenn die Datei `myfile.html` jetzt aus dem Ordner `myfolder` entfernt wird (mit Begründung)?
>
> **Lösungsvorschlag:** Problem: Aktion II wurde nicht durchgeführt, d.h. der Referenzzähler im I-Node 522 steht auf 1, obwohl zwei Hardlinks existieren. (Das Dateisystem befindet sich in einem inkonsistenten Zustand.) Wird `myfile.html` an einer Stelle entfernt, so wird der Referenzzähler um 1 dekrementiert, steht dann also auf 0. Das hat zur Folge, dass die gesamte Datei gelöscht wird, d.h. im I-Node 522 werden die Verweise auf die Blöcke 20.392 und 20.393 durch Null-Zeiger ersetzt. Im Verzeichnis `myhome` befindet sich zwar noch der Eintrag für die Datei `myfile.html`, die Datei ist aber ohne weiteres nicht mehr zugänglich (obwohl die Datenblöcke 20.392 und 20.393 möglicherweise sogar noch die korrekten Daten enthalten).

# 6 Computergestützter Entwurf von Übungsaufgaben

## 6.1 Awendungen

Wäre es nicht schön, wenn ein Roboter jedes Semester Dutzende neuer Übungsaufgaben zu allen gewünschten Vorlesungsthemen in Minutenschnelle zu Papier bringen könnte - samt Musterlösung? Fangen wir besser eine Nummer kleiner an, indem man sich zunächst die Frage stellt, für welche Arten von Aufgaben ein zumindest in Teilen automatisierter Entwurf denkbar ist. Schnell landet man dann bei Aufgaben, deren Lösungsfindung durch ein stark algorithmisches Vorgehen geprägt ist („Rechenaufgaben", wenn man so will). Sinnvoll ist der Einsatz computergestützter Entwurfskonzepte dann, wenn im Vergleich zur manuellen Erstellung eine Zeitersparnis erzielt werden kann und wenn überhaupt auch ein Bedarf nach ständig neuen Varianten dieser Aufgaben besteht, weil sie zum Beispiel als Hausaufgaben gestellt oder in Klausuren eingesetzt werden sollen. Im Zusammenhang mit der Vorlesung Informatik 3 erfüllen unter anderem die folgenden „Klassiker" diese Kriterien:

- Prozess-Scheduling
- Deadlock-Szenarien
- Merkmale von Dateisystemen
- Seitenersetzung (Paging)

Exemplarisch für die letzte Klasse von Aufgaben (Seitenersetzung) wurde im Rahmen dieser Projektarbeit eine Java-basierte Implementierung erstellt, die den Entwurf neuer Übungsaufgaben zu diesem Themenkomplex vollautomatisch übernimmt. Das lauffähige Programm (alle Java-Klassen) ist im Ordner „AutoSeitenersetzung" auf der mit dieser Ausarbeitung eingereichten CD-ROM zu finden.

## 6.2 Variable Parameter

Bei einer Übungsaufgabe zum Thema Seitenersetzung müssen verschiedene Parameter festgelegt werden. Zunächst die Rahmendaten des eigentlichen Paging-Szenarios:

- Wie viele verschiedene Seiten gibt es?
- Wie groß ist der Speicher, also aus wie vielen Seitenrahmen besteht er?
- Wie viele und welche Zugriffe sollen in der Referenzkette erfolgen?
- Soll die Referenzkette eine besonders hohe Lokalität der Speicherzugriffe widerspiegeln?
- Soll die Referenzkette so gewählt werden, dass eine FIFO-Anomalie auftritt?
- Sollen die Ergebnisse beliebige weitere formulierbare Bedingungen erfüllen (zum Beispiel: „LRU soll das beste Ergebnis liefern", „LIFO soll am schlechtesten abschneiden" oder „LFU soll genau acht Seitenfehler erzeugen")?

Diese Merkmale müssen bei der Generierung eines Szenarios zur Seitenersetzung berücksichtigt werden. Darüberhinaus gibt es einige Charakteristika, die unabhängig vom eigentlichen Szenario einfließen. Darunter fallen die folgenden Fragestellungen:

- Welche Seitenersetzungsstrategie(n) soll(en) verfolgt werden?

- Soll die Aufgabe aus zwei Teilen bestehen, in denen unterschiedliche Annahmen über die Speichergröße getroffen werden?
- Sollen Lösungsvorschläge im Multiple Choice-Stil generiert werden?

## 6.3 Spezifikation der Ausgabe

Das Programm hat die Aufgabe, aus den übergebenen Parametern selbständig eine Übungsaufgabe zu erstellen. Beispiele dafür sind Die Aufgaben 41 und 42 im Anhang. Die Ausgabe erfolgt in Form eines (LaTeX)-Dokuments, das unmittelbar in den Rahmen eines Informatik 3-Übungsblatts eingebunden werden kann. Es besteht aus

- einem einheitlicher Header mit Titel, Erstellungsdatum und Autor,
- der eigentliche Aufgabe mit Teilaufgaben und Erläuterungen und
- einer ausführlichen Musterlösung.

**Beispiel (ohne Lösung):**

```
% --------------------------------------------------------
%      Seitenersetzungs-Strategien
%      ===========================
%      File:                  H_Seitenersetzung.tex
%      Author(s):             created automatically
%      Version:               1
%      Creation Date:         Mon Oct 11 15:38:31 CEST 2004
%      Last Date of Change:   ---
%
% --------------------------------------------------------

\begin{aufgabe}{Seitenersetzungs-Strategien}{2+2+2+2}{H}

\begin{enumerate}

\item Die Menge der Seiten sei gegeben durch $N = \{0, 1, 2, 3, 4, 5\}$ und
die Menge der Seitenrahmen, die für die Speicherung der Seiten im Arbeitsspeicher
zur Verfügung steht, sei gegeben durch $Frame_3 = \{f_0, f_1, f_2, f_3\}$. Auf
die sechs Seiten der Menge $N$ werde in folgender Reihenfolge zugegriffen:

  \begin{center}
    \code{w = 2 0 4 1 5 2 0 4 3 2 0 4 1 5 3 }
  \end{center}

Ein Seitenfehler liegt immer dann vor, wenn sich eine referenzierte Seite nicht
im Arbeitsspeicher befindet. Dieser ist zu Beginn leer. Ermitteln Sie die Anzahl
der Seitenfehler für die folgenden Paging-Strategien, indem Sie alle Veränderungen
im Speicher tabellarisch dokumentieren.

  \begin{enumerate}
    \item FIFO (First In, First Out)
    \item LIFO (Last In, First Out)
    \item LRU (Least Recently Used)
```

```
  \item LFU (Least Frequently Used)
\end{enumerate}

\item  Die Menge der Seitenrahmen werde vergrößert zu $Frame_4 = \{f_0, f_1, f_2,
f_3, f_4\}$. Arbeiten Sie analog Aufgabenteil a) und vergleichen Sie die
Seitenfehlerzahlen.

\end{enumerate}

\begin{antwort}

 (...)

\end{antwort}
\end{aufgabe}
```

Eine Besonderheit liegt darin, dass eine neue Übungsaufgabe auf Wunsch auch im Multiple-Choice-Stil generiert werden kann. Zu dem modellierten Paging-Szenario werden in diesem Fall insgesamt fünf (bzw. zehn) wahre und falsche Aussagen erzeugt.

**Beispiele für MC-Statements:**

- Die gegebene Referenzkette führt bei mindestens einer Strategie zum Auftreten der Seitenfehler-Anomalie (Belady's Anomalie).
- Durch die Hinzunahme eines Seitenrahmens treten mit der LRU-Strategie zur Seitenersetzung 4 Seitenfehler weniger auf.
- Die Vergrößerung des Speichers hat keinen Einfluss auf die Seitenfehlerrate bei Anwendung der LIFO-Strategie.
- Mit der LFU-Strategie ergeben sich nun 8 Seitenfehler.
- Ist die Anzahl der Seitenrahmen größer oder gleich der maximalen Anzahl an Seiten, dann können außer beim Einlagern einer Seite in den Speicher keine Seitenfehler auftreten. Die Anzahl der Seiten ist in diesem Fall zugleich obere Schranke für die Seitenfehlerrate.

## 6.4 Bedienung: Erzeugen neuer Aufgaben

Das Programm muss nicht bedarf keiner speziellen Installation und benötigt auch keine Zusatzsoftware oder Bibliotheken. Somit sollte es auf jedem Rechner problemlos laufen, auf dem eine möglichst aktuelle Version der Java Virtual Machine zur Verfügung steht. Der Aufruf erfolgt ohne Parameter, die erforderlichen Eingaben werden vom Programm über die Konsole abgefragt. Eine graphische Oberfläche steht nicht zur Verfügung.

Der Aufruf

```
java Aufgabe
```

startet die selbsterklärende Parameterabfrage. Hier ein Beispiel:

```
Willkommen! Sie erstellen jetzt eine Uebungsaufgabe zum Thema Seitenersetzung.

Schwierigkeitsgrad der Aufgabe? (e fuer easy, m fuer medium, h fuer hard)
m

Diese Aufgabe als Multiple-Choice-Aufgabe? (j fuer ja, n fuer nein)
n

Zweite Teilaufgabe mit einem Seitenrahmen mehr? (j fuer ja, n fuer nein)
j

Soll Beladys-Anomalie bei FCFS auftreten? (j fuer ja, n fuer nein)
j

Die Werte fuer die Seitenzugriffe (referenzierte Seiten) koennen durch den Generator
unter den gewaehlten Parametern zufaellig festgelegt werden. Moechten Sie, dass die
Werte zufaellig bestimmt werden, oder wollen Sie alle Angaben manuell eingeben?
(z fuer zufaellig (empfohlen!), m fuer manuell)
m

Wie viele Seitenrahmen sollen zur Verfuegung stehen?
3

Wie viele verschiedene Seiten gibt es?
5

Wie viele Zugriffe auf den Speicher gibt es?
10

Hinweis: Da Sie die Referenkette selbst eingeben moechten, werden die bisher gemachten
Angaben zum Schwierigkeitsgrad und zur Erzwingung einer Anomalie verworfen. Beachten
Sie, dass Ihre nachfolgenden Eingaben ganze Zahlen zwischen 0 und 4 sein muessen
(jeweils einschliesslich der Grenzen).
Geben Sie jetzt bitte nacheinander die Nummern der referenzierten Seiten (fuer alle
Zugriffe) ein.

1: 0

2: 1

3: 4

4: 1

(...)

10: 3

Aufgabe wird erstellt ... Aufgabe wurde erstellt. Dateiname: H_Seitenersetzung.tex
```

**Hinweis:** Die Anzahlen der Seitenrahmen, Seiten und Zugriffe, sowie die Zugriffsfolge (Referenzkette) selbst werden nur dann abgefragt, wenn explizit eine manuelle Eingabe gewünscht wird. In diesem Fall kann das Auftreten einer FIFO-Anomalie natürlich nicht mehr garantiert werden. Im Allgemeinen dürfte eine manuelle Eingabe nicht sinnvoll sein, da bei der automatischen Generierung Aspekte berücksichtigt werden, die dazu beitragen, dass die erzeugte Referenzkette ein reales Verhalten möglichst gut abbildet (z.B. Zugriffslokalität).

## 6.5 Die Algorithmen

Für die Musterlösungen mussten alle verschiedenen Seitenersetzungsalgorithmen in Java implementiert werden. Exemplarisch hier die Umsetzung von „LFU (Least Frequently Used)“:

```
private void incPointerLfu() {
  int temp = memory2d[0][1]; framePointer = 0;

  for(int i=1; i < numOfFrames; i++) {
    if(memory2d[i][1] < temp) {
      temp = memory2d[i][1];
      framePointer = i;
    }
  }
}

private String makeAnswerLfu() {

  framePointer = 0;
  memory2d = new int[numOfFrames][2];

  for(int i=0; i < numOfFrames; i++) {
    memory2d[i][0] = -1;
    memory2d[i][1] = 0;
  }

  int[][][] a = new int[numOfAccess][numOfFrames+2][2];

  for(int i=0; i < numOfAccess; i++) {
    a[i][0][0] = access[i];
  }

  for(int i=0; i < numOfAccess; i++) {
    for(int j=1; j < numOfFrames+2; j++) {
      a[i][j][0] = -1;
      a[i][j][1] = 0;
    }
  }

  int pageFaults = 0;

  for(int i=0; i < numOfAccess; i++) {
    for(int j=1; j < numOfFrames+2; j++) {
      if(i!=0) {
        a[i][j][0] = a[i-1][j][0];
        a[i][j][1] = a[i-1][j][1];
      }
    }
```

```
    incPointerLfu();
    if(!(isInMemory2d(access[i]))) {
      pageFaults++;
      a[i][numOfFrames+1][0] = pageFaults;
      a[i][framePointer+1][0] = access[i];
      a[i][framePointer+1][1] = 1;
      memory2d[framePointer][0] = access[i];
      memory2d[framePointer][1] = 1;
    }
    else {
      int cf = 0; //cf = change frequency
      for(int n=0; n < numOfFrames; n++) {
        if(access[i] == memory2d[n][0]) {
          cf = n;
        }
      }
      a[i][cf+1][1] = a[i][cf+1][1] + 1;
      memory2d[cf][1] = memory2d[cf][1] + 1;
    }
  }

  pageFaultsLfu = pageFaults;
  String retStr = new String();

  for(int i=0; i < numOfAccess; i++) {
    retStr = retStr.concat("\n          ");
    for(int j=0; j < numOfFrames+2; j++) {
      if(j==numOfFrames+1) {
        retStr = retStr.concat(a[i][j][0] + " \\\\");
      }
      else {
        if(a[i][j][0]==-1) {
          retStr = retStr.concat("-,0 & ");
        }
        else {
          if(j==0) {
            retStr = retStr.concat(a[i][j][0] + " & ");
          }
          else {
            retStr = retStr.concat(a[i][j][0] + "," + a[i][j][1] + " & ");
          }
        }
      }
    }
  }
  return retStr;
}

public static void main(String args[]) {
  Aufgabe a = new Aufgabe("easy", false, false);
}

public int getPageFaultsFifo() {
  return pageFaultsFifo;
}
```

```
}
```

# 7 Zusammenfassung & Ausblick

Im Rahmen dieser Projektarbeit wurde der Aufgabenfundus für die Vorlesung Informatik 3 von Grund auf überarbeitet. Die Verzeichnisstruktur wurde verbessert und die bestehenden Aufgaben in die neue Ordnerhierarchie eingebettet. Etwa 50 neue Übungsaufgaben einschließlich ihrer ausführlichen Lösungen wurden erstellt, außerdem zahlreiche alte Aufgaben aktualisiert, korrigiert oder eine Musterlösung hinzugefügt.

Als mehrteilige Vertiefungsaufgabe wurde die Simulation eines Scheduling-Systems unter Umsetzung der in der Vorlesungstheorie behandelten Konzepte in der Programmiersprache Java implementiert. Aufbauend auf dem Wunsch nach stets verschieden parametrisierten Übungs- und Klausuraufgaben zu den wichtigen (Standard-)Themen wurde ein Programm erstellt, das eine Aufgabe zum Thema „Seitenersetzung" vollautomatisch erstellt.

Eine denkbare Erweiterung zu dieser Projektarbeit liegt in der Weiterverfolgung des Perl/HTML-basierten Ansatzes zur dynamischen Aufgabenvorschau (Vgl. Kapitel 4) und in der Entwicklung einer Schnittstelle zur kompletten Verwaltung des Aufgabenfundus und auch des Übungsbetriebs. Einige Aspekte:

- **Veranstaltungs-Profile:** Für bestimmte Vorlesungen (z.B. „Informatik 3 im Wintersemester 2005/2006") können Profile angelegt werden, die verschiedene Parameter speichern, wie das Semester, den Namen des Dozenten, das Erscheinungsdatum des ersten Übungsblattes, die Periodik der Übungsblätter (z.B. wöchentlich), Ferienzeiten und Ausnahmen, die Anzahl der insgesamt erscheinenden Blätter, usw.
- **Übungsblatt-Erstellung:** Mehrere Aufgaben können auf der Nutzerschnittstelle (z.B. über Checkboxen) markiert/ausgewählt und aus ihnen ein Übungsblatt erstellt werden. Im Dialog mit dem Benutzer werden weitere Informationen für dieses Blatt (Nummer, aktuelle Mitteilungen, Reihenfolge der ausgewählten Aufgaben auf dem Übungsblatt, usw.) abgefragt. Andere Parameter werden dem Veranstaltungsprofil entnommen.
- **Dateiverwaltung:** Aufgaben können direkt über die grafische Oberfläche umbenannt, verschoben (anderes Themengebiet), gelöscht oder dupliziert werden. Auch die Themengebiete können bei Bedarf angepasst werden.
- **Editor:** Im Quelltext der Übungsaufgaben kann direkt editiert werden, wodurch Aufgaben oder Lösungen korrigiert oder modifiziert werden können. Ebenso können neue Aufgaben erstellt werden, wozu eine Vorlage (Master) zur Verfügung steht.
- **Pics & Sources:** Die zu einer Aufgabe gehörenden Bilder oder Quellcodes können über das Nutzer-Interface ebenfalls verwaltet werden. So ist es jederzeit möglich, die zu einer Aufgabe gehörenden Dateien auszumachen, um Änderungen und Updates komfortabel umzusetzen. Zu neuen oder bei Änderungen an bestehenden Aufgaben können neue Grafiken und Sourcen eingepflegt werden.
- **Versionskontrolle:** Das System erkennt automatisch die aktuellste Version einer Aufgabe und kann unterschiedliche Versionen von unterschiedlichen Varianten unterscheiden. In der Standard-Ausgabe werden ältere Versionen ausgeblendet bzw. geeignet gekennzeichnet.

- **Typisierung:** Manchmal kann es sinnvoll sein, Aufgaben eines bestimmten Typs in einer anderen Rolle zu verwenden – beispielsweise aus einer P-Aufgabe eine H-Aufgabe zu machen. Das sollte möglich sein, ohne zuvor den Quelltext verändern zu müssen.
- **Nutzerverwaltung:** Unterschiedliche Personen haben unterschiedliche Privilegien und Rechte hinsichtlich der Nutzung und des Zugriffs auf das Repository. Ein Administrator (z.B. der vorlesungsbetreuende Assistent) kann verschiedene Nutzer anlegen und verwalten.
- **Multiple-Choice-Module:** Aufgaben im Multiple-Choice-Stil werden nicht länger als „Fünfer-Blöcke“ gespeichert, sondern als Einzel-Statements, denen jeweils ein oder mehrere Themenbereich(e) zugewiesen werden. Bei der Zusammenstellung einer Multiple-Choice-Aufgabe kann der Nutzer den oder die abzudeckenden Themenbereich(e) festlegen und anschließend aus einer Übersicht die gewünschten Statements/Module auswählen. Eine Datenbank- oder XML-basierte Lösung würde sich anbieten.
- **Sperren:** Idealerweise wird bei der Bearbeitung einer Datei automatisch ein Lock auf diese gelegt, sodass andere Nutzer zeitweise keinen Zugriff haben. So können Datenverlust und „schmutzige“ Updates im Mehrbenutzerbetrieb verhindert werden. In der Praxis dürfte diese Anforderung von eher untergeordneter Bedeutung sein, da wohl selten mehrere Personen unkoordiniert am Repository arbeiten werden.

# 8 Anhang: Neue Übungsaufgaben im Wintersemester 2004/2005

(Aufgaben: siehe Beiheft)

1. H_BS_Einfuehrung.tex
2. H_BS_Ebenen_V2.tex
3. H_MC_BS-Grundlagen.tex
4. H_MC_BS-Modi.tex
5. T_Betriebsmittel.tex
6. T_PowerManagementACPI.tex
7. H_MC_Powermanagement.tex
8. P_EinfacheBefehle.tex
9. P_ShellProgrammierungEinfuehrung.tex
10. P_ShellProgrammierung_2.tex
11. H_MC_Prozesse.tex
12. H_ZustandsProzessModell.tex
13. H_MC_Prozess-Zustandsmodell.tex
14. K_MultiprogrammingAuslastung_V4.tex
15. P_EinfacherThread.tex
16. H_FalscherThread.tex
17. H_MC_Multithreading.tex
18. H_MC_UserKernelLevelThreads.tex
19. H_MC_Scheduling.tex
20. K_SchedulingVermischt.tex
21. P_SchedSim_Teil1.tex
22. P_SchedSim_Teil2.tex
23. P_SchedSim_Teil3.tex
24. K_SchedSimFCFS.tex
25. T_DeadlockModellierung.tex
26. T_Prozessfortschrittsdiagramm_V2.tex
27. H_MC_Prozessfortschrittsdiagramm.tex

28. H_MC_Deadlock_V1.tex
29. H_MC_Deadlock_V2.tex
30. H_MC_Deadlock_V3.tex
31. H_DeadlockPrevention_1_V2.tex
32. H_DeadlockWiederholung_1.tex
33. K_WDH_Deadlocks.tex
34. T_RaceCondition.tex
35. H_MC_RaceCondition.tex
36. H_Zaehlsemaphore.tex
37. H_MC_ErzeugerVerbraucher.tex
38. K_JavaSynchronizedRaetsel_V2.tex
39. T_JavaMonitore_2.tex
40. T_SpeicherverwaltungUeberblick.tex
41. H_Seitenersetzung_Auto_V1.tex
42. H_Seitenersetzung_AutoMitBelady_V2.tex
43. K_SeitenersetzungLFU.tex
44. H_MC_Seitenersetzung_V1.tex
45. H_MC_Seitenersetzung_V2.tex
46. H_MC_Seitenersetzung_V3.tex
47. H_MC_SeitenersetzungsStrategien_V1.tex
48. H_MC_SeitenersetzungsStrategien_V2.tex
49. K_BuddyVerfahren.tex
50. T_InterpreterLRU_KellerAlgorithmen.tex
51. H_KellerAlgorithmen.tex
52. H_MC_SpeicherPartitionierung.tex
53. H_Speicherverwaltung_V3.tex
54. K_SpeicherVermischt.tex
55. T_VirtuellerSpeicherAdressierung.tex
56. H_EngpassIO.tex
57. T_JournalingDateisysteme.tex
58. H_Ext2Ext3Recovery.tex

59. K_WDH_JournalingDateisysteme.tex
60. K_Shellskript_HardSoftlink.tex
61. K_WDH_iNodes.tex
62. K_MC_iNodes_1.tex
63. K_MC_iNodes_2.tex
64. P_JavaTanz.tex
65. T_JavaCertExam_V2.tex
66. H_Wdh_3.tex

# Literatur

[1] Andrew S. Tanenbaum, Moderne Betriebssysteme, Pearson Studium, 2. Auflage, Januar 2002

# Abbildungsverzeichnis