

# Betriebssysteme

SS 2015

**Hans-Georg Eßer**

Dipl.-Math., Dipl.-Inform.

## **Foliensatz U:**

- Unix und Ulix
- Literate Programming
- Aufbau der Ulix-Quellcode-Dateien

v1.0, 2015/04/30  
(mit Literate Programming)

# Übersicht: BS Praxis und BS Theorie

A	Einführung	Software-Verwaltung	E
	Shell	Scheduler / Prioritäten	F
	Dateiverwaltung	Synchronisation	G
B	Filter	Speicherverwaltung	S
	C-Compiler	Dateisysteme, Zugriffsrechte	T
C	Prozesse / Jobs	<b>Einführung Ulix</b>	← Folien U
	Threads	Ulix: Interrupts, Faults	V
D	Interrupts	Zusammenfassung	H
	System Calls		

# Unix und Ulix

- Wie alle (modernen) BS bietet Unix eine Abstraktion der Hardware und ein Interface, über das Prozesse diese nutzen können
- Prozessor (Multi-Tasking, Scheduling)
- Prozessverwaltung (`fork`, `exec`, `wait`, `exit`)
- Festplatte/Diskette (Dateisystem)
- Prozess-Kommunikation und -Synchronisation
- Netzwerk: TCP/IP, Sockets
- Multi-User-Betrieb

- Unix stellt die Ressource CPU mehreren Prozessen zur Verfügung
- je nach Unix-Version verschiedene Scheduling-Strategien
- Priorisierung von Prozessen (nice values), Steuerung über `nice()` und `setpriority()`
- kein Prozess kann die CPU monopolisieren

- Prozesse bei Unix in Baumstruktur organisiert („Vater/Sohn“)
- Neue Prozesse werden mit `fork ( )` als ident. Kopie des aufrufenden Prozesses erzeugt
- Prozess kann mit `exec ( )` ein anderes Programm nachladen
- `exit ( )` beendet Prozess (mit Rückgabewert)
- `wait ( )`, `waitpid ( )` warten auf Ende eines Kindprozesses, Auswerten des Rückgabewerts

- Intern verwaltet Unix Prozesse über Prozess-Kontroll-Blöcke (PCBs)
- jeder Prozess hat eigenen Adressraum (Speicherschutz), der zudem vom Adressraum des Unix-Kernels getrennt ist
- traditionell: keine Threads (nur über User-Level-Bibliothek)
- Context Switch wechselt von einem Prozess zum nächsten (Scheduler entscheidet, wann und zu welchem Prozess)

- Alle Unix-Dateisysteme haben gemeinsame Eigenschaften (→ Foliensatz T)
  - hierarchisches Dateisystem (Unterordner)
  - Medien über Mountpoints in Baum integriert
  - Inodes verwalten Eigenschaften und Blockliste einer Datei
  - Inodes enthalten *keinen* Dateinamen
  - Verzeichnisse sind spezielle Dateien mit Zuordnungen  
Dateiname → Inode-Nummer
  - freie Datenblöcke und freie Inodes in Bitmaps verwaltet
  - Superblock enthält Verwaltungsinformationen für gesamtes Dateisystem

- System Calls für Zugriff auf Dateien
  - `fd=open ( )` öffnet Datei, gibt file descriptor zurück
  - `read (fd, ...)` liest aus Datei
  - `write (fd, ...)` schreibt in Datei
  - `lseek (fd, ...)` springt zu Position in Datei
  - `close (fd)` schließt Datei
- Interne Umsetzung oft über Virtual Filesystem (VFS)
- unterstützt mehrere FS und diverse Hardware

- Signal-Mechanismus
  - Signal-Handler, `signal()`
  - Signal senden, `kill()`
- Auch BS selbst verwendet Signale
- Synchronisation über Mutexe und/oder Semaphore

- TCP/IP (Verbindungen), UDP (verbindungslos)
- Sockets (Client/Server)
- Verwendung von Sockets ähnlich wie Zugriff auf Dateien,
  - `socket()`,  
TCP: `SOCK_STREAM`,  
UDP: `SOCK_DGRAM`
  - I/O multiplexing mit `select()`

- Unix kennt mehrere Benutzer, root (UID=0) ist Systemverwalter
- Anmeldung über Username, Passwort
- Benutzergruppen
- Dateizugriff: Zugriffsrechte (für Besitzer, Gruppe, sonstige)
- Signalversand: nur an eigene Prozesse

UNIX 0.12 ist erste fertige Release.

- **Prozesse:**

- ✓ fork, exit, wait, exec funktionieren;  
RR-Scheduler

- ✗ aktuelle Version ohne Prioritäten (→ BA, fertig)

- **Dateisystem:**

- ✓ VFS, Hardware: Disketten, IDE-Platten,  
Minix-v2-FS

- **Signale:**
  - ✓ Datenstrukturen im PCB vorhanden, `signal()` trägt Handler ein, `kill()` verschickt Signal
  - ✓ Kernel passt Stack des Empfängerprozess an, damit er beim nächsten Aufruf den Signal Handler ausführt
- **Speicher:**
  - ✓ Paging: Seitentabellen für jeden Prozess, Kernel-Adressraum wird beim Wechsel in Kernel-Mode sichtbar;  
Auslagern / Wiedereinlagern von Seiten

- **System Calls:**

- ✓ System Call Interface, erlaubt Registrieren neuer Syscall-Handler.  
Standard-Syscalls für alle Kernel-Funktionen vorhanden  
User-Mode-Bibliothek enthält Funktionen, welche diese Syscalls aufrufen

- **Interrupts:**

- ✓ Interrupt-Handler für Timer, FDC, IDE, Tastatur, serielle Schnittstelle

- **Zugriff auf Geräte:**

- ✓ Floppy: DMA, Block-Transfer
- ✓ Festplatte: PIO, Block-Transfer
- ✓ „Serial Harddisk“: byte-weiser Transfer über serielle Schnittstelle
- ✓ Gerätedateien (für Blockgeräte)

- **Netzwerk:**

- ✗ nicht implementiert (in BA: IP via SLIP)

- **Multi-User-Betrieb:**

- ✓ alle nötigen Datenstrukturen vorhanden
- ✓ Zugriffsrechte im Minix-Dateisystem
- ✓ Anmeldung (Benutzer/Passwort), Benutzerwechsel mit `su`

- **Terminals:**

- ✓ UNIX startet Shells auf mehreren Terminals
- ✗ kein Grafikmodus; Terminals laufen in 80x24-Textmodus (Zeile 25: BS-Statuszeile)

```
$ for i in bin-build/ulix.c bin-build/start.asm lib-build/ulixlib.[ch]; do \
  echo -n "$i: "; grep -v "^#line" $i | wc -l; done
bin-build/ulix.c:          7510
bin-build/start.asm:      241
lib-build/ulixlib.c:      517
lib-build/ulixlib.h:      376
```

```
Ulix-i386 0.12                               Build: Fr  5 Dez 2014 23:10:47 CET
Current time: 2015/04/30 12:03:31
RAM: 64 MByte, mapped to 0xD0000000-0xD3FFFFFF
UT:  Initialized ten terminals (press [Alt-1] to [Alt-0])
FDC: fd0 (1440 KByte), fd1 (1440 KByte)
ATA: hda (1440 KByte), hdb (100000 KByte)
mount: dev[03:00] = /dev/hda on /           type minix (options 0)
mount: dev[02:01] = /dev/fd1 on /mnt/      type minix (options 0)
mount: dev[03:40] = /dev/hdb on /tmp/     type minix (options 0)
mount: none           on /dev/           type dev   (options 0)
swapon: enabling /tmp/swap (64 MByte)
Starting five shells on tty0..tty4. Press [Ctrl-L] for de/en keyboard.
swapper launched in background. output at console 10

ulix login: esser
Password: (auto login)
esser@ulix[6]:/home/esser$ _

Ulix-i386 0.12  SCH:002  IRQ:0N                tty0  FF=3b1e  AS=0001  12:03:48
```

# Literate Programming

- LaTeX (gespr.: La-Tech) ist eine Auszeichnungssprache für Dokumente, grob vergleichbar mit HTML
- Zwei wesentliche Syntax-Elemente:
  - Befehl:  
`\befe1{argument1}{argument2}...`
  - Umgebung:  
`\begin{umgebung}`  
`...`  
`\end{umgebung}`

- „Hello-World“-Dokument:

```

\documentclass{article}           % versch. Klassen, z. B.
                                  % book, report, letter
\begin{document}                 % Anfang des Dokuments
  Hello, World.                  % „body“
\end{document}                   % Ende des Dokuments

```

- Überschriften

```

\section{Überschrift 1}           % Stufe 1
Text

```

```

\subsection{Überschrift 1.1}     % Stufe 2
Text

```

```

\subsubsection{Überschrift 1.1.1} % Stufe 3

```

```

% testdatei1.tex

\documentclass{article}
\usepackage[latin1]{inputenc}

\begin{document}

\tableofcontents % Inhaltsverz.

\section{Beispiel}

Das ist ein kleines Beispiel.

\subsection{Unterabschnitt}

Mehr im Beispieldokument \dots

\subsubsection{Überschrift 1.1.1}

So geht es \emph{kursiv} und
\textbf{fett}.

\end{document}

```

```

$ pdflatex testdatei1.tex
$ pdflatex testdatei1.tex

```

→ erzeugt testdatei1.pdf:

## Contents

1	Beispiel	1
1.1	Unterabschnitt . . . . .	1
1.1.1	Überschrift 1.1.1 . . .	1

## 1 Beispiel

Das ist ein kleines Beispiel.

### 1.1 Unterabschnitt

Mehr im Beispieldokument ...

#### 1.1.1 Überschrift 1.1.1

So geht es *kursiv* und **fett**.

```

\documentclass{article}
\usepackage[latin1]{inputenc}

\begin{document}

\tableofcontents    % Inhaltsverz.

\section{Beispiel}

Das ist ein kleines Beispiel.

\subsection{Unterabschnitt}

Mehr im Beispieldokument \dots

\subsubsection{Überschrift 1.1.1}

So geht es \emph{kursiv} und
\textbf{fett}.

\end{document}

```

```

<html>
<head><meta charset="iso-8859-1"></head>

<body>

<!-- Inhalt gibt es nicht -->

<h1>1. Beispiel</h1>

Das ist ein kleines Beispiel.

<h2>1.1 Unterabschnitt</h2>

Mehr im Beispieldokument .. .

<h3>1.1.1 Überschrift 1.1.1</h3>

So geht es <i>kursiv</i> und
<b>fett</b>.

</body>
</html>

```

```
\begin{enumerate}
  % Aufzählung (1, 2, 3...)
  \item Erster
  \item Zweiter
  \item Dritter
\end{enumerate}
```

```
\begin{itemize}
  % Auflistung (Bullets)

  \item Erster
  \item Zweiter
  \item Dritter
\end{itemize}
```

```
\includegraphics[width=
  10cm]{bild.png}
```

```
<ol>
  <!-- Ordered List -->
  <li> Erster </li>
  <li> Zweiter </li>
  <li> Dritter </li>
</ol>
```

```
<ul>
  <!-- Unordered List,
        bullets -->
  <li> Erster </li>
  <li> Zweiter </li>
  <li> Dritter </li>
</ul>
```

```

```

1. Erster
2. Zweiter
3. Dritter

- Erster
- Zweiter
- Dritter

(Bild)

Erster Absatz. Text im ersten Absatz.

Text im zweiten Absatz. Text im zweiten Absatz. Text im zweiten Absatz. Text im zweiten Absatz. Text im zweiten Absatz. Text im zweiten Absatz.

Text im dritten Absatz. Text im dritten Absatz. Text im dritten Absatz. Text im dritten Absatz. Text im dritten Absatz.

Harter Umbruch \\  
 Neue Zeile \\  
 Noch eine Zeile

`<p>` Erster Absatz. Text im ersten Absatz. `</p>`

`<p>` Text im zweiten Absatz. `</p>`

`<p>` Text im dritten Absatz. `</p>`

`<p>` Harter Umbruch `<br>`  
 Neue Zeile `<br>`  
 Noch eine Zeile`</p>`

- `pdflatex`: wandelt `tex`-Datei in `pdf`-Datei um
  - ggf. mehrfach aufrufen (z. B. für Inhalt)
  - bei Fehlern: [Eingabe] drücken
  - wiederholte Fehler: X eingeben (und [Eingabe] drücken)
- `xpdf`: PDF-Betrachter (Evince, Gnome-App)
- `bibtex`: hilft bei der Literaturverwaltung
  - verwendet `bib`-Datei
  - ggf. später mehr dazu

```
ulix@ulixdevel:~/tex$ pdflatex test2.tex
This is pdfTeX, Version 3.1415926-1.40.10 (TeX Live 2009/Debian)
entering extended mode
(./test2.tex
LaTeX2e <2009/09/24>
Babel <v3.81> and hyphenation patterns for english,
usenglishmax, dumylang, nohyphenation, loaded.
(/usr/share/texmf-texlive/tex/latex/base/article.cls
Document Class: article 2007/10/19 v1.4h Standard LaTeX document
class
(/usr/share/texmf-texlive/tex/latex/base/size10.clo))
No file test2.aux.
[1{/var/lib/texmf/fonts/map/pdftex/updmap/pdftex.map}]
(./test2.aux) )</usr/sha
re/texmf-texlive/fonts/type1/public/amsfonts/cm/cmr10.pfb>
Output written on test2.pdf (1 page, 12056 bytes).
Transcript written on test2.log.
```

```
ulix@ulixdevel:~/tex$ ls -l test2.*
-rw-r--r-- 1 ulix ulix    79 13. Okt 22:39 test2.aux
-rw-r--r-- 1 ulix ulix  2488 13. Okt 22:39 test2.log
-rw-r--r-- 1 ulix ulix 32965 13. Okt 22:39 test2.pdf
-rw-r--r-- 1 ulix ulix   112 13. Okt 22:39 test2.tex
-rw-r--r-- 1 ulix ulix    53 13. Okt 22:39 test2.toc
```

- test2.log: Protokoll, mit Warnungen
- test2.pdf: Erstellte PDF-Datei
- test2.toc: Daten für Inhaltsverzeichnis  
(toc = table of contents)

```

ulix@ulixdevel:~/tex$ pdflatex test1.tex
This is pdfTeX, Version 3.1415926-1.40.10 (TeX Live 2009/Debian)
entering extended mode
(./test1.tex
LaTeX2e <2009/09/24>
Babel <v3.81> and hyphenation patterns for english, usenglishmax,
dumylang, nohyphenation, loaded.
(/usr/share/texmf-texlive/tex/latex/base/article.cls
Document Class: article 2007/10/19 v1.4h Standard LaTeX document class
(/usr/share/texmf-texlive/tex/latex/base/size10.clo)) (./test1.aux)

! LaTeX Error: Environment empyhd undefined.

See the LaTeX manual or LaTeX Companion for explanation.
Type H <return> for immediate help.
...

1.4 \begin{empyhd}

? X
No pages of output.
Transcript written on test1.log.

```

- Grundidee: Nicht Code kommentieren, sondern „erzählen“, wie das Programm funktioniert
- Code und Dokumentation vertauscht
- Beispiel für den Einstieg: Bubblesort  
(Original-Code von <http://de.wikipedia.org/wiki/Bubblesort> übernommen; nach C portiert)

```
void bubblesort() {
    int SIZE = 10;
    int i, j, tmp;
    for (j=SIZE; j>1; j--) {
        for (i=0; i<j-1; i++) {
            if (arr[i] > arr[i+1]) {
                tmp = arr[i+1];
                arr[i+1] = arr[i];
                arr[i] = tmp;
            }
        }
    }
}
```

# Bubblesort (2): mit Kommentaren

```
// bubblesort: sort elements of an array arr[] of size SIZE
void bubblesort() {
    // declarations
    int SIZE = 10;    // size of the array
    int i, j;        // loop variables
    int tmp;         // temporary variable for swapping elements
    // outer loop
    for (j=SIZE; j>1; j--) {
        // check all neighbors in arr[0..j-1] and swap them if their
        // order is wrong
        for (i=0; i<j-1; i++) {
            if (arr[i] > arr[i+1]) {
                // swap neighbors i, i+1 (using tmp as temporary variable)
                tmp = arr[i+1];
                arr[i+1] = arr[i];
                arr[i] = tmp;
            }
        }
    }
}
```

## Bubblesort: The Literate Program

To sort a field `arr[]`, we first need to initialize a `SIZE` variable and declare some local variables, such as `i` and `j` which are used as loop counters, as well as a temporary variable `tmp`:

```
< bubblesort: declarations 15 > ≡  
int SIZE = 10;  
int i, j, tmp;
```

The main routine of the bubblesort algorithm compares each field element with its direct (right) neighbor and corrects their order if it is wrong. After doing this once the biggest element will be at the end of the list. It then repeats these steps with a smaller field (ignoring the right-most element). Thus, with each step in a loop, the unsorted array becomes one element smaller until there is nothing left to sort:

```
< bubblesort program 16 > ≡  
void bubblesort() {  
    < bubblesort: declarations 15 >  
    for (j=SIZE; j>1; j--) {  
        < bubblesort: check neighbors in range 0..j 17 >  
    }  
}
```

In order to do the neighbor checks, a second loop inside the outer loop is necessary:

```
< bubblesort: check neighbors in range 0..j 17 > ≡  
for (i=0; i<j-1; i++) {  
    if (arr[i] > arr[i+1]) {  
        < bubblesort: swap elements 17 >  
    }  
}
```

For swapping, three commands and usage of a temporary variable are necessary in a C program:

```
< bubblesort: swap elements 17 > ≡  
tmp = arr[i+1];  
arr[i+1] = arr[i];  
arr[i] = tmp;
```

- Elemente der Form *<name>* heißen **Code Chunks**.
- Code Chunks werden, wie Makros, an den Stellen ersetzt, wo sie auftreten, z. B.

```
void bubblesort() {  
    <bubblesort: declarations 15>  
    for (j=SIZE; j>1; j--) {  
        ...  
    }
```



```
void bubblesort() {  
    int SIZE = 10;  
    int i, j, tmp;  
    for (j=SIZE; j>1; j--) {  
        ...  
    }
```

- Code Chunks dürfen nicht rekursiv sein:

```
< unfug 19 > ≡  
for (i=0; i<10; i++) {  
    < unfug 19 >  
}
```



- Chunk kann vor oder nach der Benutzung definiert werden
- Chunk darf auch komplett fehlen  
→ verursacht nur eine Warnung

- Code Chunks können an mehreren Stellen definiert werden
- spätere Auftreten setzen die Definition fort

```
< lange definition 20 >≡
for (i=0; i<10; i++) {
    // erste Schleife
}
```

Text, andere Chunks, ...

```
< lange definition 20 >+≡
for (i=0; i<10; i++) {
    // zweite Schleife
}
```



wird später zu:

```
for (i=0; i<10; i++) {
    // erste Schleife
}
for (i=0; i<10; i++) {
    // zweite Schleife
}
```

**≡ vs. +≡ !**

## Syntax in LaTeX-Dokumenten:

- `<<chunk name>>` für *`<chunk name>`*
- Definition: `<<name>>= ... @`
- `[[variable]]` für variable

```
<<bubblesort program>>=
void bubblesort() {
  <<bubblesort: declarations>>
  for (j=SIZE; j>1; j--) {
    <<bubblesort: check neighbors in range 0..[[j]]>>
  }
}
@
```

## Komplettes Literate Program in LaTeX/NoWeb:

```
\subsubsection{Bubblesort: The Literate Program}
```

To sort a field, we first need to initialize a `[[SIZE]]` variable and declare some local variables, such as `[[i]]` and `[[j]]` which are used as loop counters, as well as a temporary variable `[[tmp]]`:

```
<<bubblesort: declarations>>=  
int SIZE = 10;  
int i, j, tmp;  
@
```

The main routine of the bubblesort algorithm compares each field element with its direct (right) neighbor and corrects their order if it is wrong. After doing this once the biggest element will be at the end of the list. It then repeats these steps with a smaller field (ignoring the right-most element). Thus, with each step in a loop, the unsorted array becomes one element smaller until there is nothing left to sort:

```

<<bubblesort program>>=
void bubblesort() {
  <<bubblesort: declarations>>
  for (j=SIZE; j>1; j--) {
    <<bubblesort: check neighbors in range 0..[[j]]>>
  }
}
@

```

In order to do the neighbor checks, a second loop inside the outer loop is necessary:

```

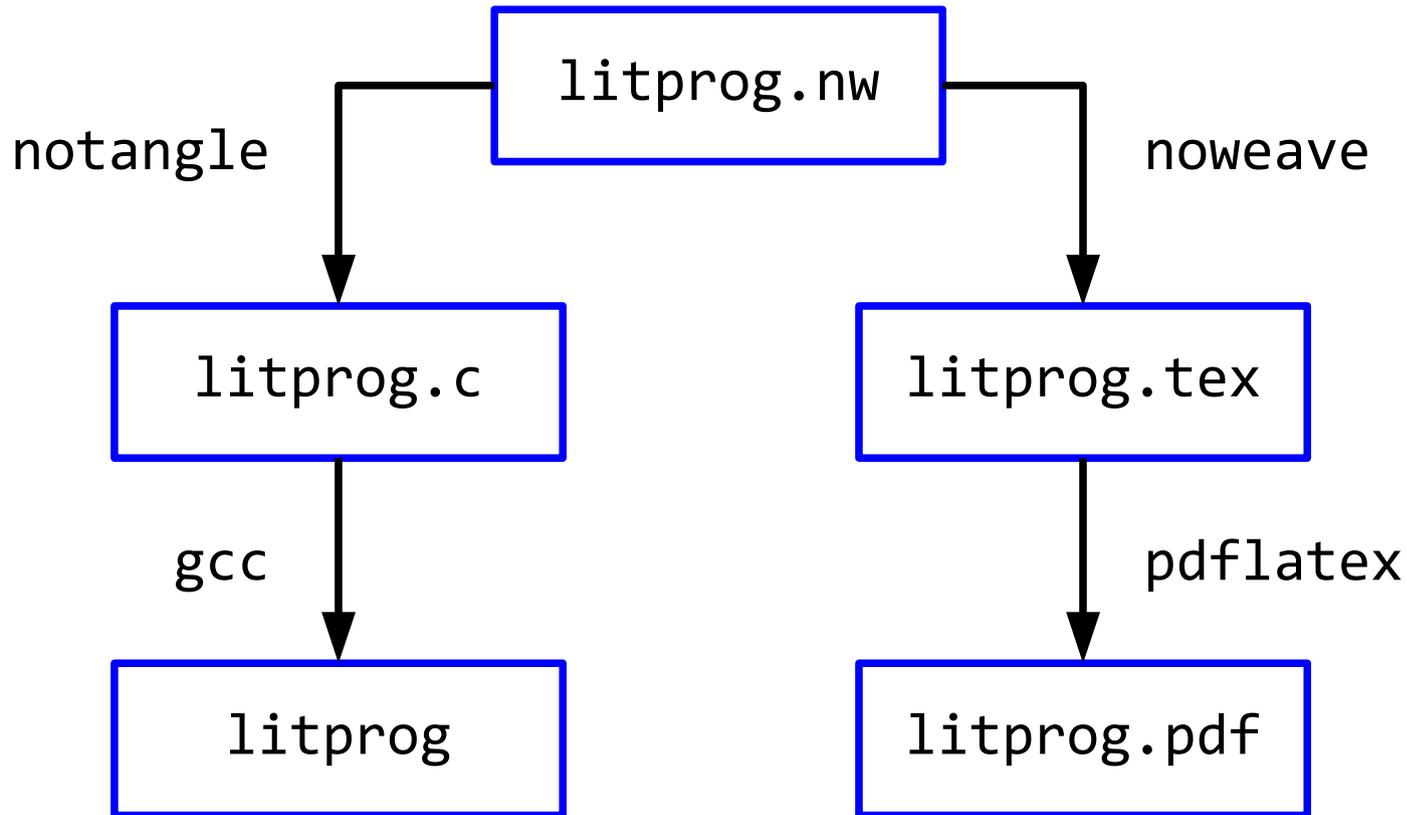
<<bubblesort: check neighbors in range 0..[[j]]>>=
for (i=0; i<j-1; i++) {
  if (arr[i] > arr[i+1]) {
    <<bubblesort: swap elements>>
  }
}
@

```

For swapping, three commands and usage of a temporary variable are necessary in a C program:

```
<<bubblesort: swap elements>>=  
tmp = arr[i+1];  
arr[i+1] = arr[i];  
arr[i] = tmp;  
@
```

- NoWeb-Tools: noweave, notangle



- Syntax von noweave:  
`noweave datei.nw > datei.tex`
- nw-Datei enthält immer ein vollständiges Literate Program

- Syntax von notangle:  
`notangle -Rchunkname datei.nw > code.c`
- nw-Datei enthält viele Code Chunks – welchen Sie extrahieren wollen, legen Sie mit der Option `-Rchunkname` fest.
- Zusatzoption `-L`: gibt Hinweise zu Zeilennummern in nw-Datei aus (für den Compiler)

- Auszug aus `ulix/bin-build/Makefile`:

```
TEXSRC_FILE=../ulix-book.nw  
TEXSRC_MODULE_FILE=../student.nw
```

extract:

```
notangle -L -Rulix.c $(TEXSRC_FILE) > ulix.c; true  
notangle -L -Rprintf.c $(TEXSRC_FILE) > printf.c  
notangle -Rstart.asm $(TEXSRC_FILE) > start.asm  
notangle -Rulix.ld $(TEXSRC_FILE) > ulix.ld  
notangle -L -Rmodule.c $(TEXSRC_MODULE_FILE) > module.c  
notangle -L -Rmodule.h $(TEXSRC_MODULE_FILE) > module.h
```

(Zeilennummern mit `-L`; nur für C-Dateien)

*„In der Informatik bezeichnet man einen Entwicklungsprozess für Software als **Top-down**, wenn der Entwurf mit abstrahierten Objekten beginnt, die dann konkretisiert werden; der Prozess ist **Bottom-up**, wenn von einzelnen Detail-Aufgaben ausgegangen wird, die zur Erledigung übergeordneter Prozesse benötigt werden.“*

(Quelle: [http://de.wikipedia.org/wiki/Top-down\\_und\\_Bottom-up](http://de.wikipedia.org/wiki/Top-down_und_Bottom-up))

- Literate Programming unterstützt beide Ansätze und auch Misch-Varianten
- Reihenfolge der Präsentation (der Code Chunks) legt fest, ob der Code top-down oder bottom-up entwickelt wird
- Beispiel: Betriebssystem

- Ulix muss den Speicher und die Platte initialisieren und dann die Shell von Platte laden und starten:

```
<<ulix>>=  
  <<initialize memory>>  
  <<initialize harddisk>>  
  <<load shell program from disk>>  
  <<run shell>>
```

@

- Wie kann man nun den Speicher initialisieren?

```
<<initialize memory>>=  
  <<check available memory>>  
  <<create initial page table>>
```

@

- Ulix muss zunächst den Speicher initialisieren; dafür braucht es eine Seitentabelle:

```
<<page table declaration>>=  
typedef struct {  
    unsigned int present           : 1; // 0  
    unsigned int writeable        : 1; // 1  
    ...  
    unsigned int frame_addr       : 20; // 31..12  
} page_desc;  
  
typedef struct {  
    page_desc pds[1024];  
} page_table;  
@
```

# Aufbau des Ulix-Codes

*<ulix.c 28a>* ≡  
*<copyright notice 8>*  
*<constants 96a>* *<public constants 30a>*  
*<macro definitions 19a>* *<public macro definitions 560a>*  
*<public elementary type definitions 29e>*  
*<type definitions 75>* *<public type definitions 126a>*  
*<function prototypes 29a>* *<public function prototypes 418b>*  
*<global variables 76b>*  
*<function implementations 84b>* *<public function implementations 419a>*  
*<kernel main 28b>* 

```
<kernel main 28b>≡  
void main () {  
    <initialize kernel global variables 152d>  
    <setup serial port 329a> // for debugging  
    <setup memory 81>  
    <setup video 321c>  
    <setup keyboard 302e>  
    <initialize system 29b> ←  
    <initialize syscalls 157d>  
    <initialize filesystem 29c>  
    <initialize swap 277b>  
    initialize_module (); // external code  
    <start init process 29d>  
}
```

$\langle \textit{initialize system 29b} \rangle \equiv$   
 $\langle \textit{install the interrupt descriptor table 130e} \rangle$   
 $\langle \textit{install the fault handlers 132b} \rangle$   
 $\langle \textit{install the interrupt handlers 123b} \rangle$   
 $\langle \textit{install the timer 323a} \rangle$   
 $\langle \textit{enable interrupts 31b} \rangle$



Das schauen wir uns  
 im Detail an!  
 → Foliensatz F (neu)

$\langle \textit{initialize system 29b} \rangle + \equiv$

... (u. a.: blocked queues initialisieren, Paging aktivieren etc.)