Design, Implementation and Evaluation of the ULIX Teaching Operating System

Hans-Georg Eßer

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Lehrstuhl für IT-Sicherheitsinfrastruktur (Informatik I)

23.03.2015

Übersicht

- *Orientierung*: Betriebssysteme (BS), BS-Vorlesungen
- Beiträge der Arbeit
- ULIX: ein Unix-ähnliches Lehr-BS
- *Orientierung*: Literate Programming
- ULIX-Buch: ein BS-Lehrbuch
- ULIX-Vorlesung: Kursaufbau, Präsentationstool, Evaluation
- Zusammenfassung

Betriebssysteme

- Kernfach in den Informatik-Curricula (bei technisch orientierten Studiengängen)
- wesentliche Grundlagen-Inhalte über die Jahrzehnte relativ konstant (Prozesse, Synchronisation, Speicher, Interrupts und Fehler, Dateisystem und I/O, System Calls, Netzwerke, IPC)
- Verwaltung der Rechner-Ressourcen
  - löst Konflikte bei gleichzeitiger Ress.-Anforderung
  - vereinfacht Hardware-Nutzung durch Interfaces
  - abstrahiert die Maschine (Virtualisierung)

Lehr-Betriebssysteme

- gleiche Prinzipien wie „richtige“ BS
- Fokus auf grundlegende Datenstrukturen, Algorithmen, Strategien
  - keine Optimierung
  - eingeschränkter Hardware-Support
  - wenige Features, wenige Anwendungen

BS-Kurse an Hochschulen

- Fokus auf Einführungsveranstaltungen
- Vorlesungen i. d. R. sehr theoretisch
- praktische Anteile:
  - oft: „Papier-und-Stift“-Aufgaben zur Theorie
  - oft: Systemprogrammierung
  - manchmal: Implementierungsprojekte an kleinen Lehr-BS
  - selten: Implementierungsprojekte an „echten“ BS, z. B. Linux (zu komplex für Erstkontakt / ein Semester)

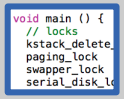
Ziele

- Lehr-BS entwickeln, mit dem Studenten an konkretem Code die Funktionsweise nachvollziehen können
- BS-Lehrbuch schreiben, das die Präsentation dieses Codes integriert
- zugänglichere Präsentation als in bisherigen Arbeiten (z. B. *Minix*, *xv6*, *Xinu*: Theorietexte mit Code-Anhang oder Code-Ausschnitten)
- Evaluation

Beiträge der Arbeit

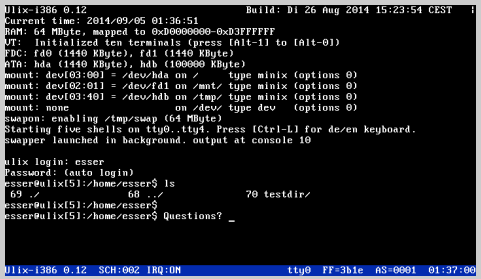
Design, Implementation and Evaluation of the ULIX Teaching Operating System

I.



ULIX – das Betriebssytem

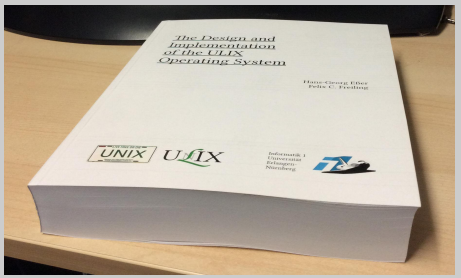
- Klassisches Unix
- Threads
- VFS



II.

ULIX-Buch – das Lehrbuch

- ~ 700 Seiten
- BS-Theorie + Code
- Literate Program



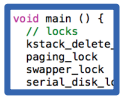
III.

ULIX-Vorlesung und Evaluierung

- Foliensätze
- Übungsaufgaben
- Verständnistests

All questions					
	absolute		scaled		
	pre-test	post-test	pre-test	post-test	improv.
ULIX course	11.71	10.00	100.00 %	85.40 %	14.60 %
Microcontrollers	13.17	10.83	100.00 %	82.23 %	17.77 %
ULIX-course-related questions					
	absolute		scaled		
	pre-test	post-test	pre-test	post-test	improv.
ULIX course	8.36	6.71	100.00 %	80.26 %	19.74 %
Microcontrollers	9.17	7.17	100.00 %	78.19 %	21.81 %
Ratio of ULIX-course-related / overall improvements					
ULIX course					135.16 %
Microcontrollers					122.75 %

Design-Kriterien ULIX / ULIX-Buch



- zentrale Features eines Unix-Systems
- „so einfach wie möglich“ (keine Optimierungen)
- Aufbau des Buchs an Entwicklungsreihenfolge orientiert

  → Beschreibung des Entwicklungs-*Prozess*

UNIX- vs. ULIX-Struktur

UNIX System V Release 2/3

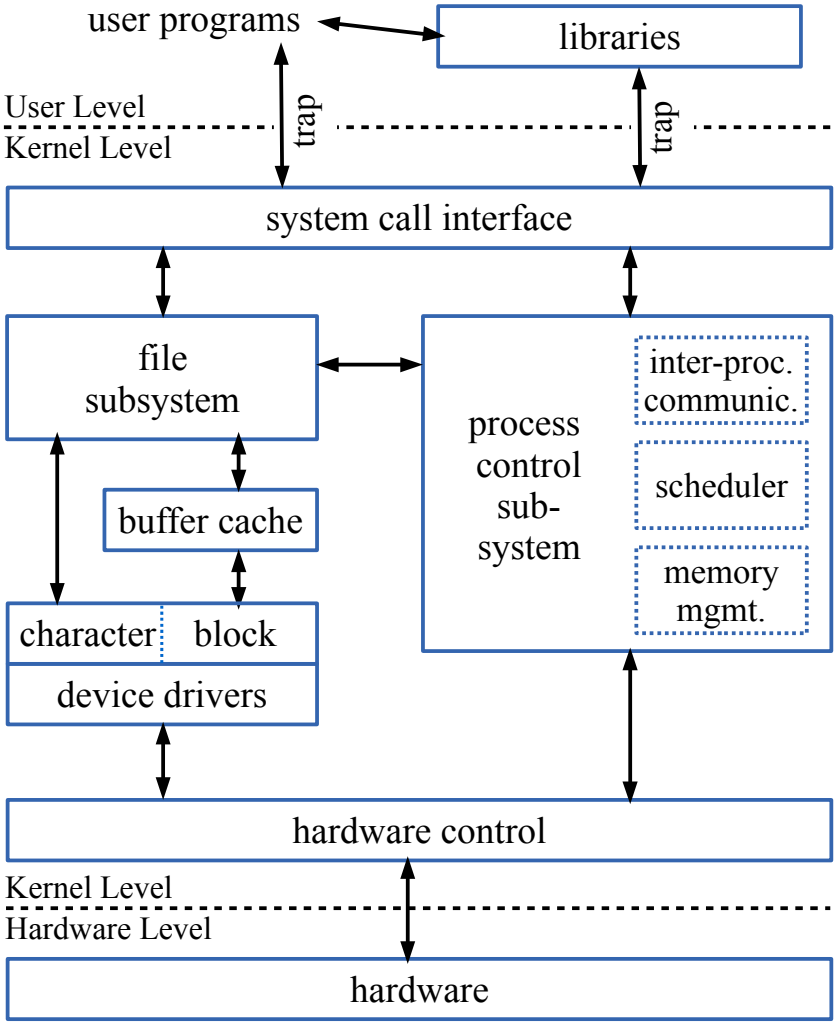
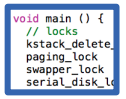


Bild: nach M. J. Bach 1986 [Bac86]

UNIX- vs. ULIX-Struktur



UNIX System V Release 2/3

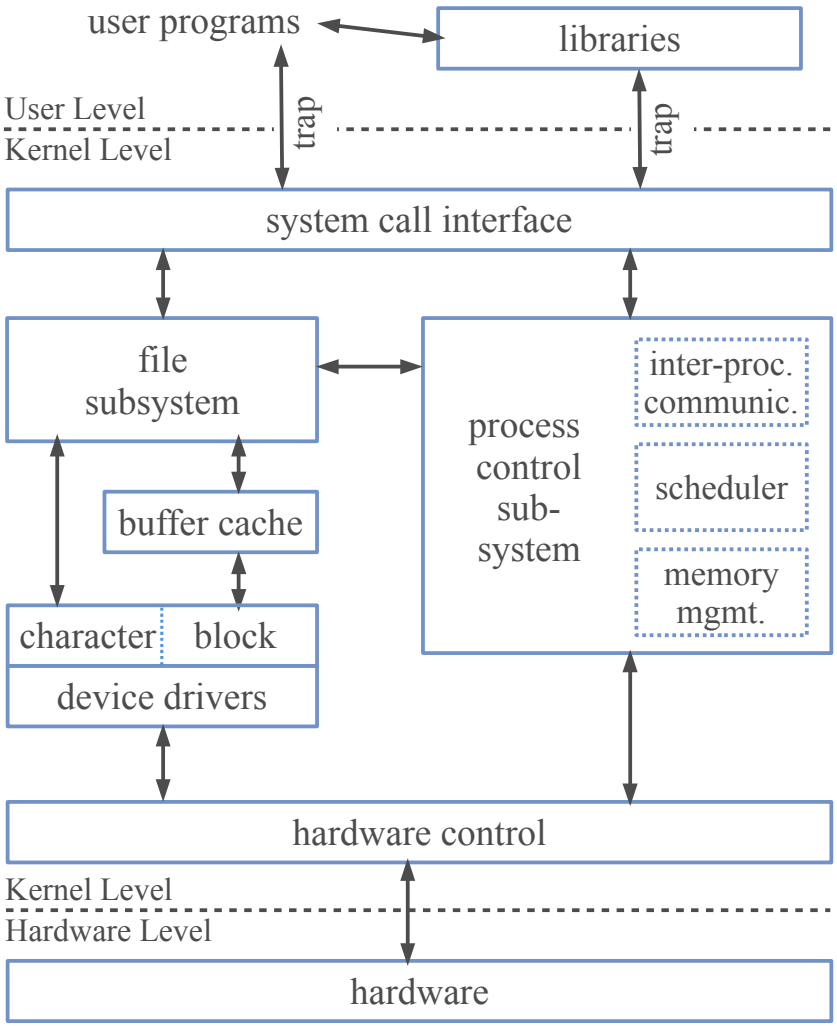
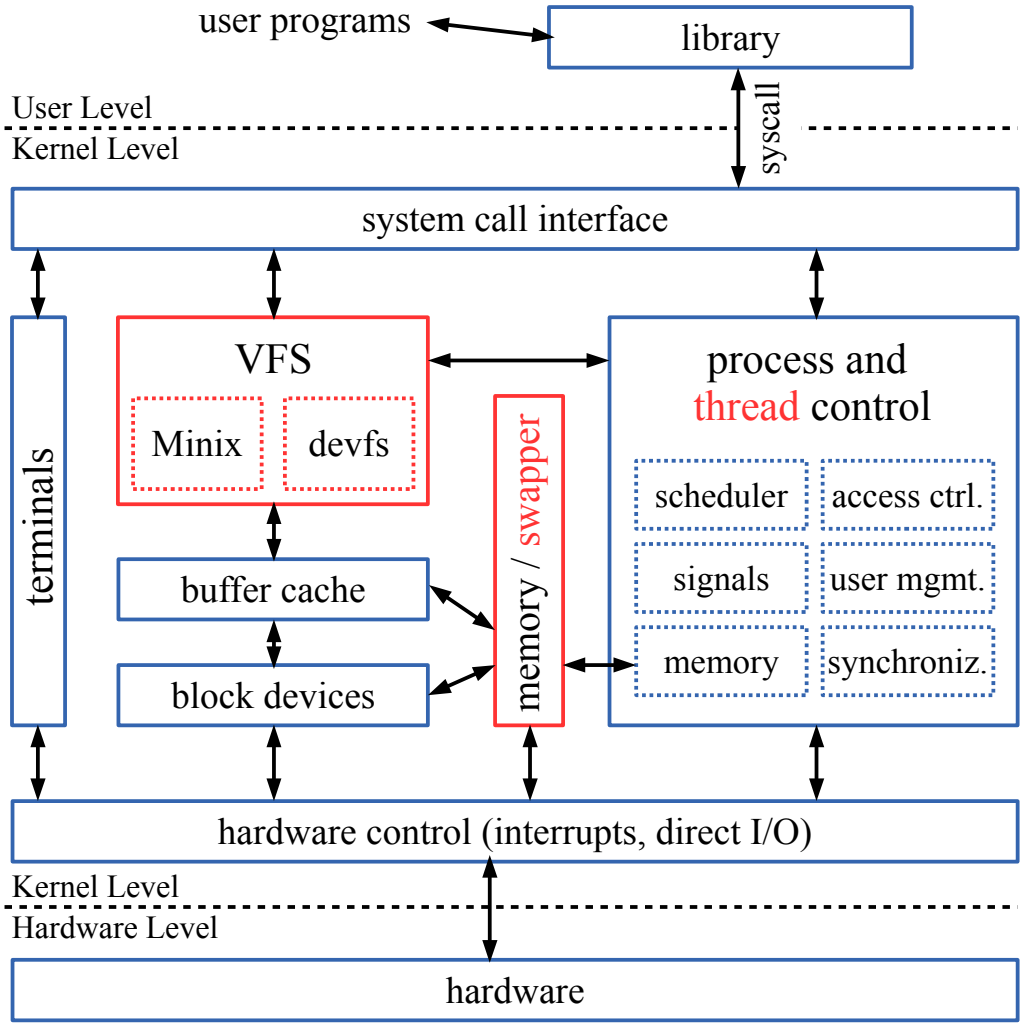


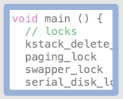
Bild: nach M. J. Bach 1986 [Bac86]

ULIX 0.12



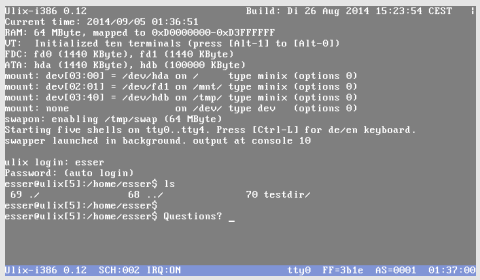
II. ULIX-Buch

I.



ULIX – das Betriebssytem

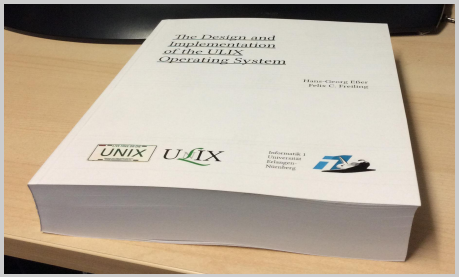
- Klassisches Unix
- Threads
- VFS



II.

ULIX-Buch – das Lehrbuch

- ~ 700 Seiten
- BS-Theorie + Code
- Literate Program



III.

ULIX-Vorlesung und Evaluierung

- Foliensätze
- Übungsaufgaben
- Verständnistests

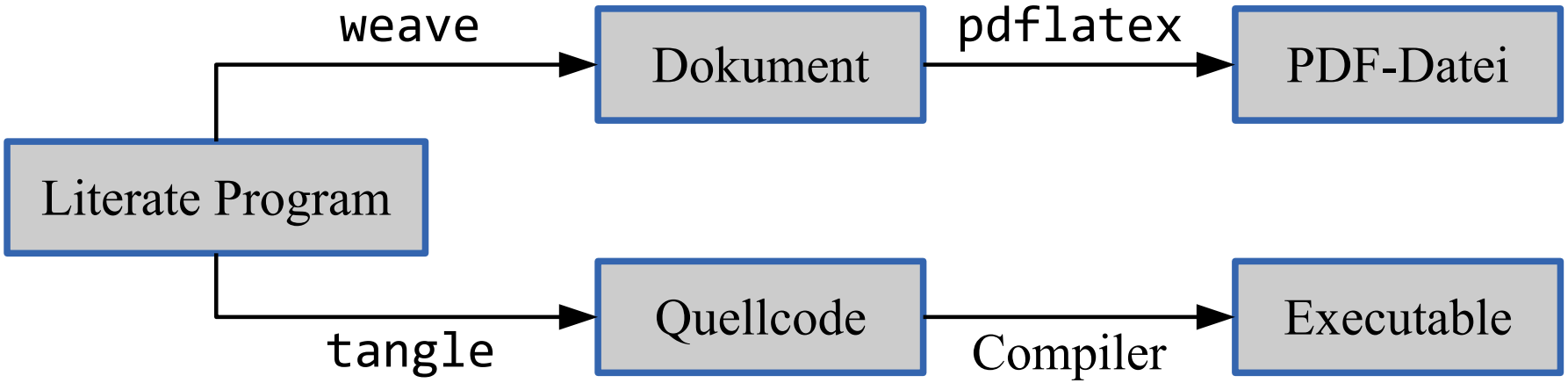
All questions					
	absolute		scaled		
	pre-test	post-test	pre-test	post-test	improv.
Ulix course	11.71	10.00	100.00 %	85.40 %	14.60 %
Microcontrollers	13.17	10.83	100.00 %	82.23 %	17.77 %
Ulix-course-related questions					
	absolute		scaled		
	pre-test	post-test	pre-test	post-test	improv.
Ulix course	8.36	6.71	100.00 %	80.26 %	19.74 %
Microcontrollers	9.17	7.17	100.00 %	78.19 %	21.81 %
Ratio of Ulix-course-related / overall improvements					
Ulix course					135.16 %
Microcontrollers					122.75 %

Literate Programming (LP)

- Donald E. Knuth: WEB (→ TEX)
- Code und Dokumentation in einem Dokument
- Programm wie ein Artikel / Buch lesbar
- benannte Code Chunks, ⟨ *Beispiel-Code* ⟩
- unterstützt Bottom-up-, Top-down- und gemischte Entwicklung

Literate Programming (LP)

- Code und Dokumentation aus derselben Quelldatei (einem WEB) extrahieren



Literate Programming: in ULIX

```
// definition of thread control block (30 fields)

typedef struct {
  thread_id       pid;         // process id
  thread_id       tid;         // thread id
  thread_id       ppid;        // parent process
  int             state;       // state of process
  context_t       regs;        // context
  memaddress      esp0;        // kernel stack pointer
  memaddress      eip;         // program counter
  memaddress      ebp;         // base pointer
  addr_space_id   addr_space;  // memory configuration
  thread_id       next;        // ``next'' thread
  thread_id       prev;        // ``previous'' thread
  boolean         used;        // entry used?
  int             error;
  int             exitcode;
  int             waitfor;
  char            cmdline[CMDLINE_LENGTH];
  boolean         new;         // new process?
  void            *top_of_thread_kstack;
  int             terminal;
  int             files[MAX_PFD];
  char            cwd[256];    // working directory
  sighandler_t    sighandlers[32];
  unsigned long   sig_pending;
  unsigned long   sig_blocked;
  word            uid;         // user ID
  word            gid;         // group ID
  word            euid;        // effective user ID
  word            egid;        // effective group ID
  word            ruid;        // real user ID
  word            rgid;        // real group ID
} TCB;
```

Literate Programming: in ULIX

```
// definition of thread control block (30 fields)

typedef struct {
  thread_id      pid;         // process id
  thread_id      tid;         // thread id
  thread_id      ppid;        // parent process
  int            state;       // state of process
  context_t      regs;        // context
  memaddress     esp0;        // kernel stack pointer
  memaddress     eip;         // program counter
  memaddress     ebp;         // base pointer
  addr_space_id  addr_space;  // memory configuration
  thread_id      next;        // ``next'' thread
  thread_id      prev;        // ``previous'' thread
  boolean        used;        // entry used?
  int            error;
  int            exitcode;
  int            waitfor;
  char           cmdline[CMDLINE_LENGTH];
  boolean        new;         // new process?
  void           *top_of_thread_kstack;
  int            terminal;
  int            files[MAX_PFD];
  char           cwd[256];    // working directory
  sighandler_t   sighandlers[32];
  unsigned long  sig_pending;
  unsigned long  sig_blocked;
  word           uid;         // user ID
  word           gid;         // group ID
  word           euid;        // effective user ID
  word           egid;        // effective group ID
  word           ruid;        // real user ID
  word           rgid;        // real group ID
} TCB;
```

So here is the declaration of the TCB structure. We have entries for the thread the process ID `pid`, a parent process ID `ppid` (so we can build a process tree), the p context (of type $\texttt{context\_t}_{126a}$) consisting of the general purpose registers and the purpose registers and a reference to the address space in which the thread runs (w already added to the data structure when we discussed address spaces). We also place for three memory addresses which hold the instruction pointer, stack poi base pointer contents). More entries will follow later.

⟨*public type definitions* 126a⟩+≡ (28a 32a) ◁12

```
typedef struct {
  thread_id  pid;         // process id
  thread_id  tid;         // thread id
  thread_id  ppid;        // parent process
  int        state;       // state of the process
  context_t  regs;        // context
  memaddress esp0;        // kernel stack pointer
  memaddress eip;         // program counter
  memaddress ebp;         // base pointer
  ⟨more TCB entries 142c⟩
} TCB;
```

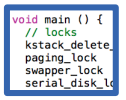
...

and 32 bits fit precisely in an `unsigned long` integer, so we can add two variables sig and `sig_blocked` for storing those bits:

⟨*more TCB entries* 142c⟩+≡ (159) ◁3

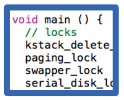
```
sighandler_t sighandlers[32];
unsigned long sig_pending;
unsigned long sig_blocked;
```

Hypothese



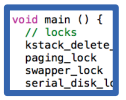

Literate Programming ist für die Präsentation des komplexen Code eines Betriebssystems und zugehörige (Programmier-) Übungen gut geeignet und kann das Verständnis der theoretischen Grundlagen wirksamer verbessern als bestehende Ansätze.

Hypothese – Komponenten




- kompletter Code eines funktional vollständigen BS mit LP in einem Lehrbuch typischer Länge dokumentierbar?
- vollständiger Code im Rahmen einer Vorlesung mit Übungen behandelbar?
- Theorie-Verständnis verbessert?
- Besser als andere Kurse?

Hypothese – Komponenten




- kompletter Code eines funktional vollständigen BS mit LP in einem Lehrbuch typischer Länge dokumentierbar? ✓ 692 Seiten
- vollständiger Code im Rahmen einer Vorlesung mit Übungen behandelbar? (✓) reduziert, u. a. ohne Dateisysteme
- Theorie-Verständnis verbessert?
- Besser als andere Kurse?

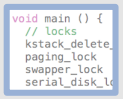
} → Evaluation

Frühere Arbeiten

- diverse (theoretische) Lehrbücher
- mehrere Lehr-BS
- einige LP-Beispiele, wenige mit komplexem Code (u. a. Textsatzsystem TeX, C-Compiler)
- zwei Ansätze zur Entwicklung eines BS mit LP (pk, Off++; beide abgebrochen)

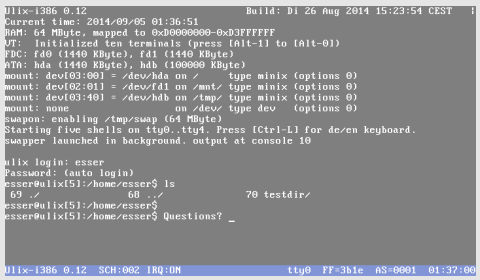
III. ULIX-Vorlesung und Evaluierung

I.



ULIX – das Betriebssytem

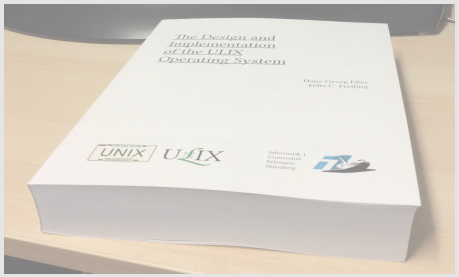
- Klassisches Unix
- Threads
- VFS



II.

ULIX-Buch – das Lehrbuch

- ~ 700 Seiten
- BS-Theorie + Code
- Literate Program



III.

ULIX-Vorlesung und Evaluierung

- Foliensätze
- Übungsaufgaben
- Verständnistests

All questions					
	absolute		scaled		
	pre-test	post-test	pre-test	post-test	improv.
ULIX course	11.71	10.00	100.00 %	85.40 %	14.60 %
Microcontrollers	13.17	10.83	100.00 %	82.23 %	17.77 %
ULIX-course-related questions					
	absolute		scaled		
	pre-test	post-test	pre-test	post-test	improv.
ULIX course	8.36	6.71	100.00 %	80.26 %	19.74 %
Microcontrollers	9.17	7.17	100.00 %	78.19 %	21.81 %
Ratio of ULIX-course-related / overall improvements					
ULIX course					135.16 %
Microcontrollers					122.75 %

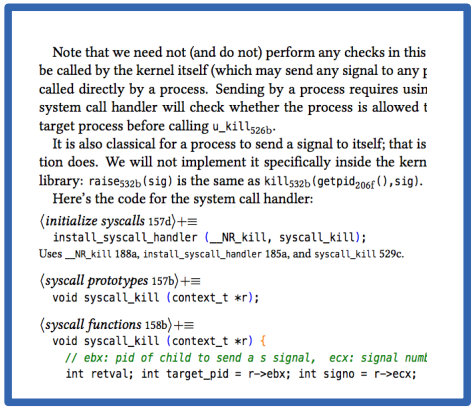
Ulix-Vorlesung und -Übungen

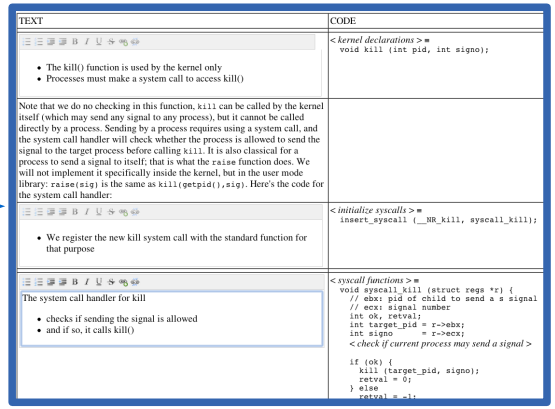
- Kurs: „Betriebssystem-Entwicklung mit Literate Programming“, 4 SWS, TH Nürnberg (WS 2013/14)
  - 2 SWS Vorlesung: Präsentation / Diskussion von ULIX-Code (→ *Literate Teaching*)
  - 2 SWS Übung: Teilnehmer ergänzen ULIX-Code (verwenden dazu auch LP)

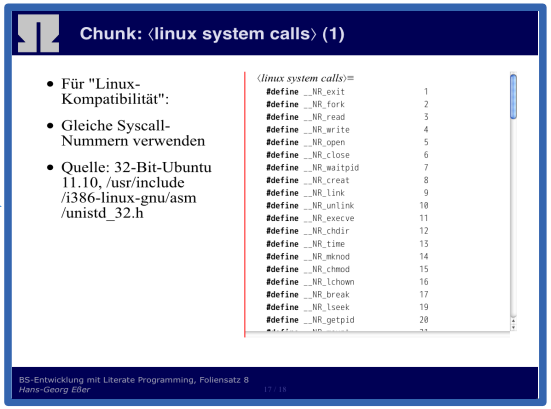
    BS wird von Woche zu Woche komplexer
  - Projektaufgabe als Abschluss (größere Aufgabe)
- basiert auf ULIX-Buch

Literate Teaching mit LiPPGen

- Erstellen eines Foliensatzes:
  - Code + Dokumentation schreiben (Buch, Artikel etc.)
  - Ausschnitt auswählen
  - mit LiPPGen Folien generieren, Folienbeschreibung ergänzen







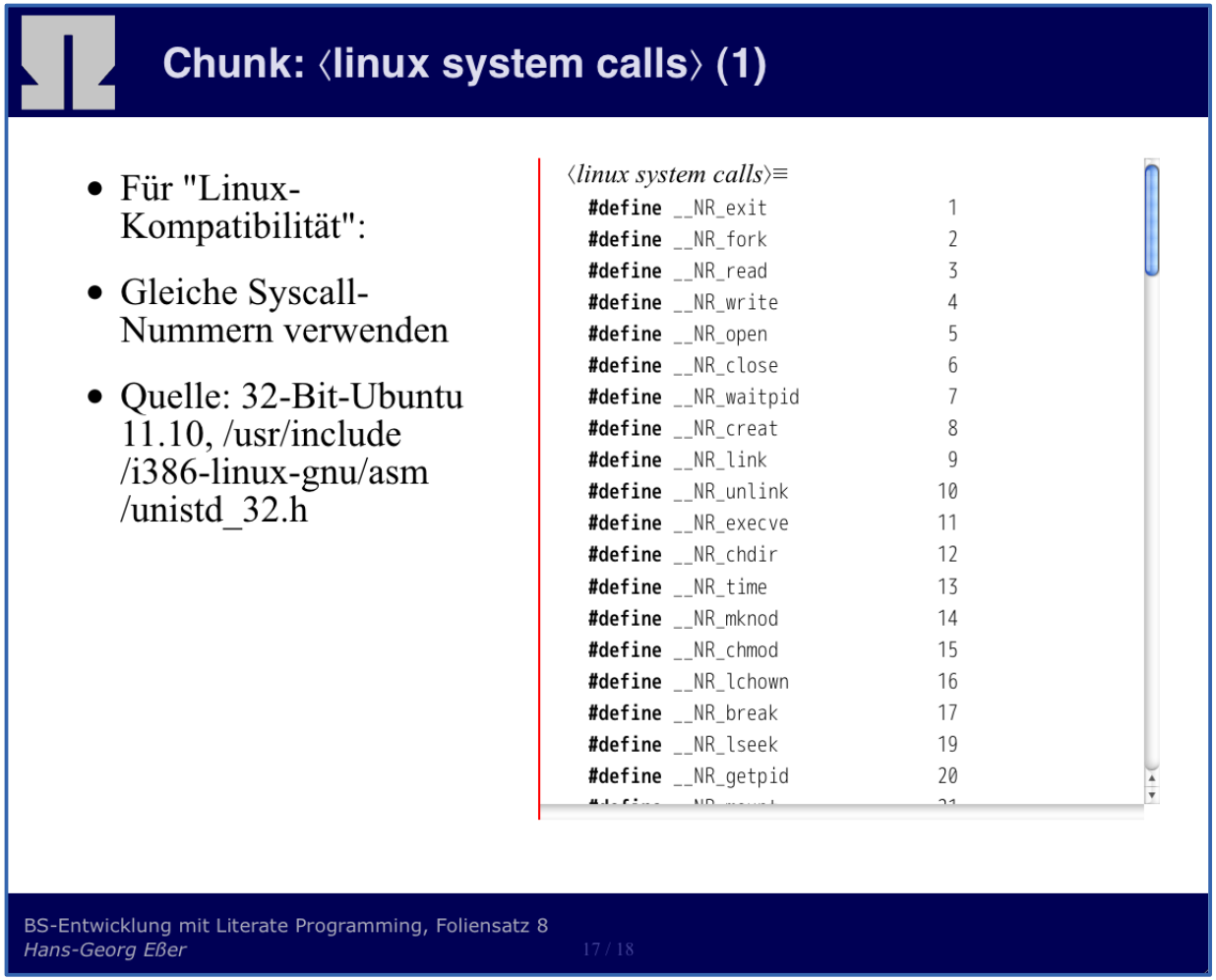
Literate Teaching mit LiPPGen

TEXT	CODE
• The kill() function is used by the kernel only • Processes must make a system call to access kill()	< *kernel declarations* > ≡ `void kill (int pid, int signo);`
Note that we do no checking in this function, `kill` can be called by the kernel itself (which may send any signal to any process), but it cannot be called directly by a process. Sending by a process requires using a system call, and the system call handler will check whether the process is allowed to send the signal to the target process before calling `kill`. It is also classical for a process to send a signal to itself; that is what the `raise` function does. We will not implement it specifically inside the kernel, but in the user mode library: `raise(sig)` is the same as `kill(getpid(),sig)`. Here's the code for the system call handler:	
• We register the new kill system call with the standard function for that purpose	< *initialize syscalls* > ≡ `insert_syscall (__NR_kill, syscall_kill);`

The system call handler for kill
• checks if sending the signal is allowed
• and if so, it calls kill()
	< *syscall functions* > ≡
	`void syscall_kill (struct regs *r) {`
	`  // ebx: pid of child to send a s signal`
	`  // ecx: signal number`
	`  int ok, retval;`
	`  int target_pid = r->ebx;`
	`  int signo      = r->ecx;`
	`  < check if current process may send a signal >`
	
	`  if (ok) {`
	`  kill (target_pid, signo);`
	`  retval = 0;`
	`  } else`
	`  retval = -1;`

- LiPPGen unterstützt bei der Erstellung der Folien
- Texte aus LP-Dokumentation in Vorschau

Literate Teaching mit LiPPGen



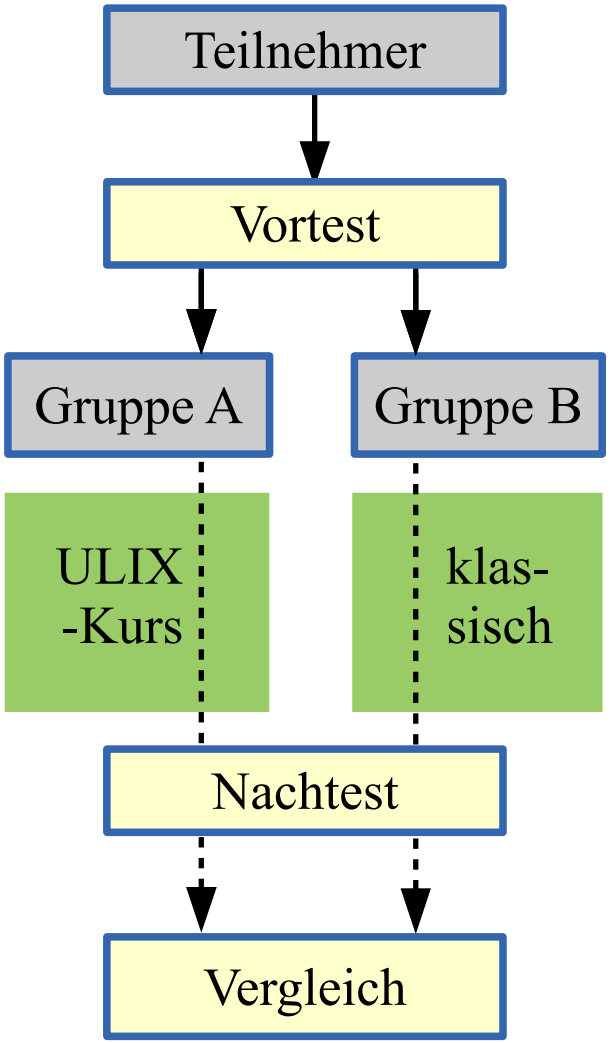
links:

- HTML
- klassische Bullet-Lists
- Bilder

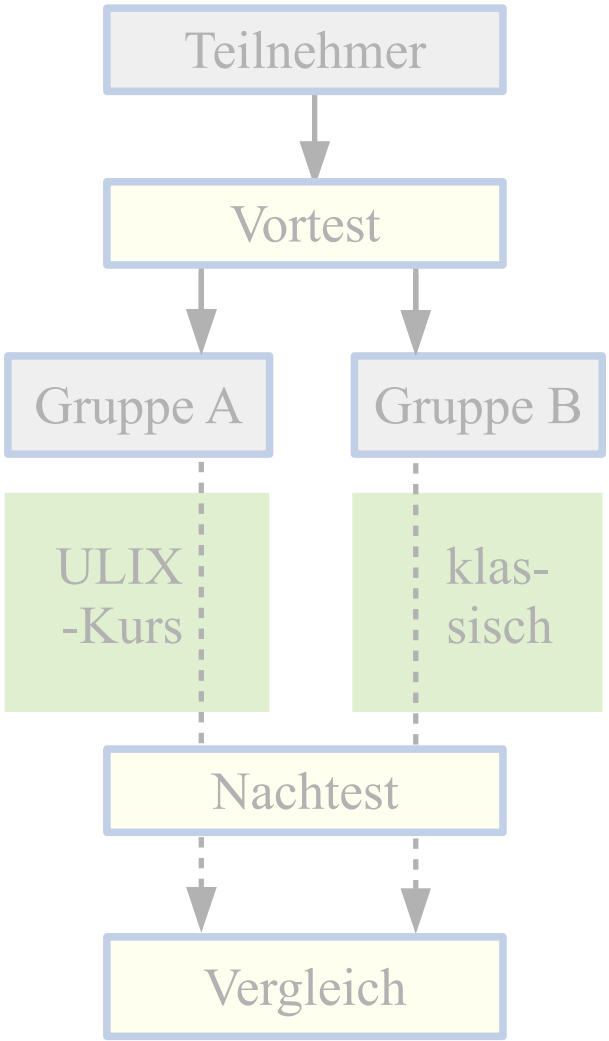
rechts:

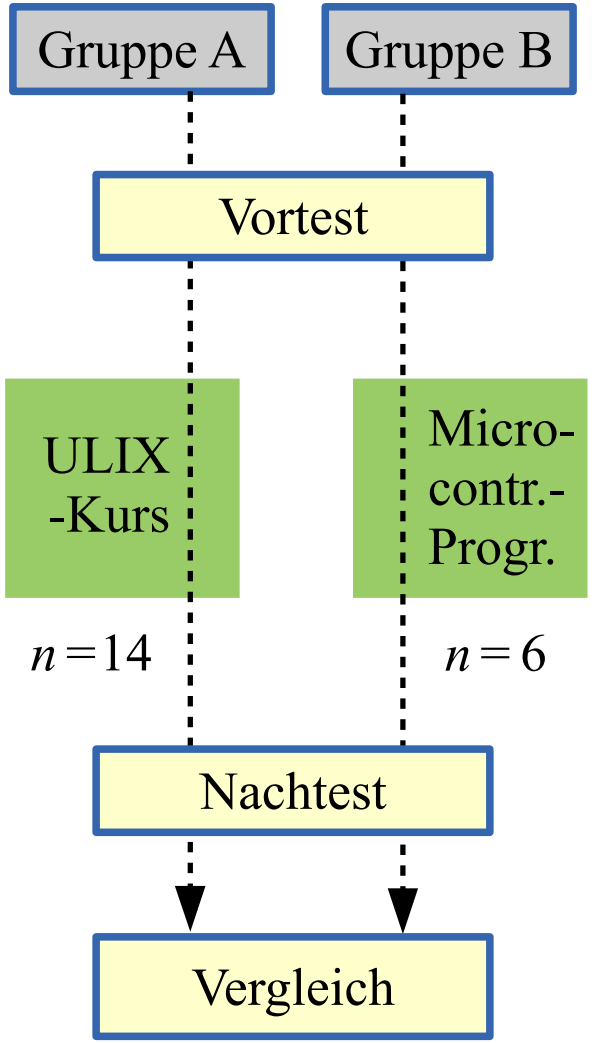
- Code-Ansicht
- scrollbar
- Syntax Highlighting
- LP-Chunks

Evaluierung

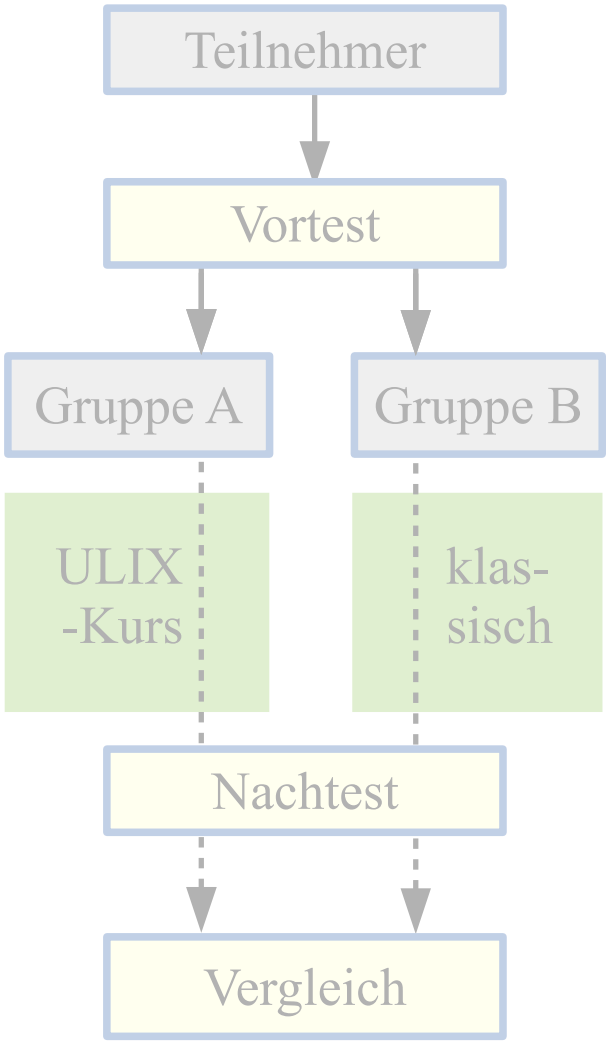


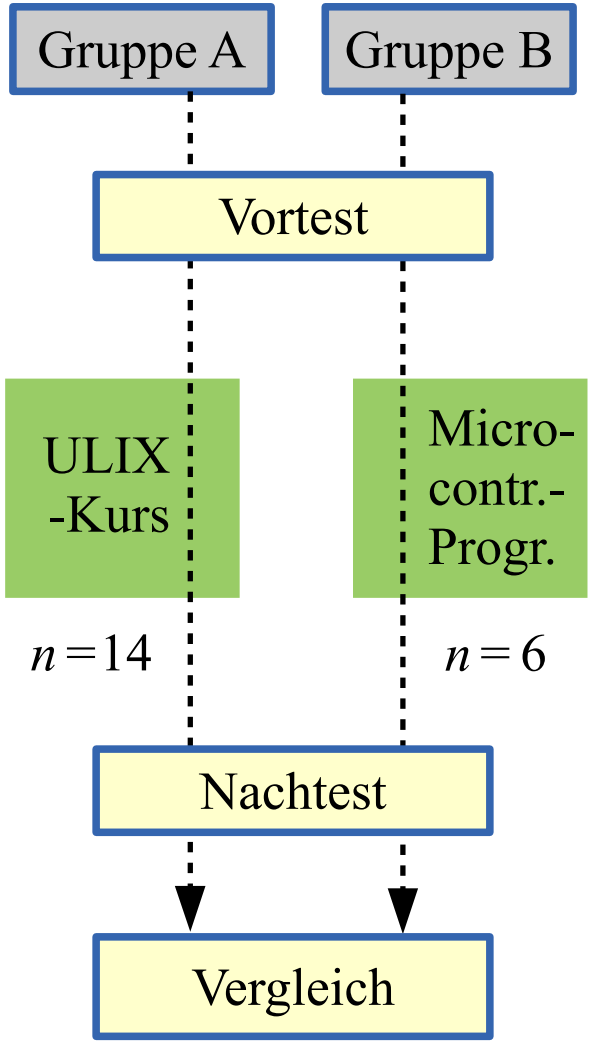
Evaluierung

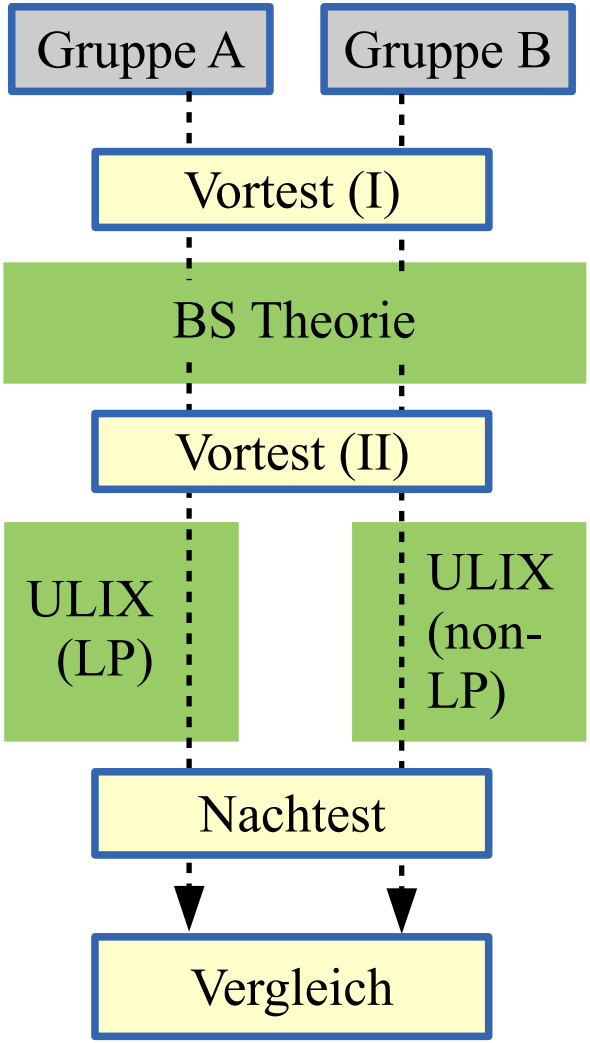




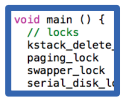
Evaluierung



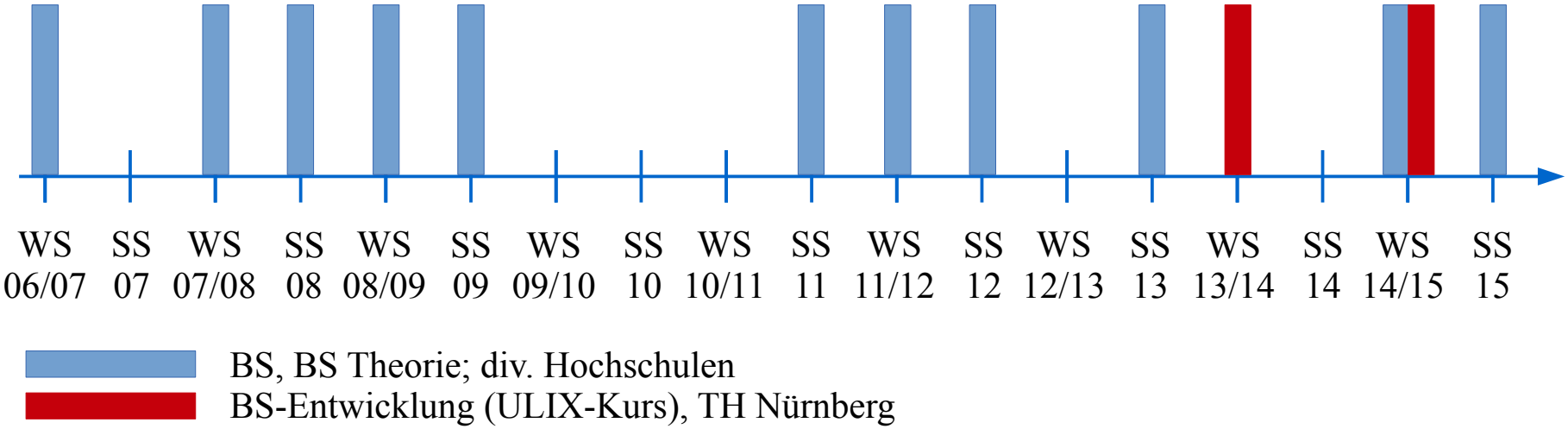




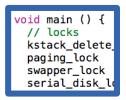
Erkenntnisse




- LP-Stil verinnerlichen → anfangs mühsam
- LP gut für „Start bei 0“
- für Vorlesung: Arbeit mit echtem Anschauungs-objekt vs. rein theoretische Betrachtungen



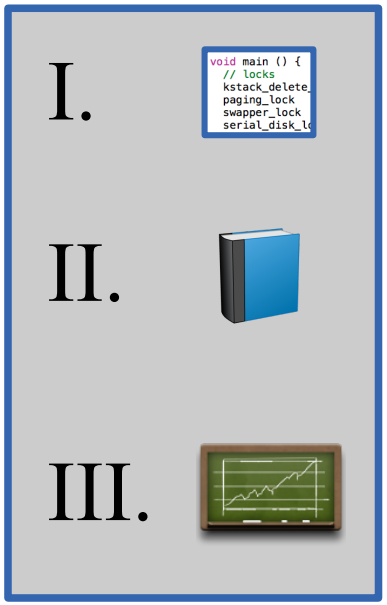
Erfahrungen aus der Lehre




- BS-Entwicklung
  - Teilnehmer aktiver als in theoretischen Vorlesungen
- Literate Programming
  - anfangs Widerstände, später (teilweise) Aha-Effekt
  - Motivation zu dokumentieren: mit LP deutlich höher
  - Oft nachgefragt: Entwicklungsumgebung
  - Teilnehmer haben eher Probleme mit C als mit LP
  - LP oft in der Form „fake literate programming“

Zusammenfassung

- Design und Implementation von ULIX
- Dokumentation in 700-seitiges BS-Lehrbuch integriert
- Konzeption, Durchführung und Evaluation einer 4-SWS-Veranstaltung



- Erstes vollständig mit LP implementiertes BS
- Erstes BS-Lehrbuch, das kompletten BS-Quellcode in reguläre Kapitel integriert


SyncMaster 2263uw
Ulix-i386 0.12 Build: Di 26 Aug 2014 15:23:54 CEST
Current time: 2014/09/05 01:36:51
RAM: 64 MByte, mapped to 0xD0000000-0xD3FFFFFF
VT: 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[5]:/home/esser$ ls
69 ./ 68 ../ 70 testdir/
esser@ulix[5]:/home/esser$
esser@ulix[5]:/home/esser$ Questions? _
Ulix-i386 0.12 SCH:002 IRQ:ON tty0 FF=3b1e AS=0001 01:37:00
ULIX TERMINAL