Konfigurierbare Systemsoftware (KSS)

VL 1 – Einführung

Daniel Lohmann

Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität
Erlangen-Nürnberg

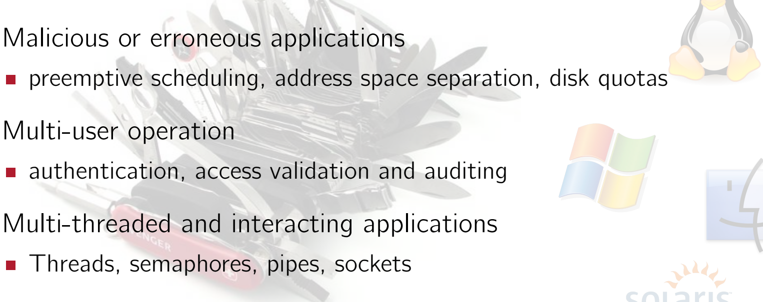
SS 16 – 2016-04-11

http://www4.informatik.uni-erlangen.de/Lehre/SS16/V_KSS

Agenda

The Operating System – A Swiss Army Knife?

Commodity operating systems provide a rich set of features to be prepared for all kinds of applications and contingencies:



- Malicious or erroneous applications
  - preemptive scheduling, address space separation, disk quotas
- Multi-user operation
  - authentication, access validation and auditing
- Multi-threaded and interacting applications
  - Threads, semaphores, pipes, sockets
- Many/large concurrently running applications
  - virtual memory, swapping, working sets

The Operating System – A Swiss Army Knife?

One size fits all? ↪ Variability

*"Clearly, the operating system design must be strongly influenced by the type of use for which the machine is intended. Unfortunately it is often the case with 'general purpose machines' that the type of use cannot be easily identified; a common criticism of many systems is that in attempting to be all things to all men they wind up being **totally satisfactory to no-one**."*

Lister and Eager 1993: *Fundamentals of Operating Systems* [4]

 $\notin$ 

The Operating System – A Swiss Army Knife?

Big is beautiful? ↪ Granularity

> *“Some applications may require only a subset of services or features that other applications need. These ’less demanding’ applications should **not be forced to pay** for the resources consumed by unneeded features.”*
>
> Parnas 1979: *“Designing Software for Ease of Extension and Contraction”* [8]

$\not\subseteq$

Variability and Granularity

Variability (Definition 1)

Variability of system software is the property that denotes the *range* of functional requirements that can be fulfilled by it.

Granularity (Definition 2)

Granularity of system software is the property that denotes the *resolution* of which requirements can be fulfilled by it, in the sense that requirements are fulfilled but not overfulfilled.

- Can general purpose (GP) systems fulfill these demands?
- Reality check – a small study with `printf()` from `glibc`:
  (Analogy: GP operating system ⟷ GP library ⟷ GP function)

```
int main() {
  printf( "Hello World\n");
}
```

Agenda

Reality Check: Granularity

- The setup:

```
> uname -a
Linux faui48a 2.6.32-5-amd64 #1 SMP Mon Oct 3 05:45:56 UTC 2011 x86_64 GNU/Linux
> gcc -dumpversion
4.4.5
```

- Experiment 1: `printf()`

```
> echo 'main(){printf("Hello World\n");}' | gcc -xc - -w -Os -static -o hello1
> ./hello1
Hello World
> size hello1
   text    data     bss     dec     hex filename
 508723    1928    7052  517703   7e647 hello1
```

512 KiB!

- Maybe the general-purpose `printf()` is just to powerful?
  - supports many data types, formatting rules, ...
  - implementation requires a complex parser for the format string
- Let’s try the much more specialized `puts()`!

Reality Check: Granularity (Cont'd)

- Experiment 2: `puts()`

```
> echo 'main(){puts("Hello World");}' | gcc -xc - -Os -w -static -o hello2
> ./hello2
Hello World
> size hello2
   text    data     bss     dec     hex filename
 508723    1928    7052  517703   7e647 hello2
```

512 KiB!

- That didn't help much!
- Maybe `puts()` is yet too powerful?
  - buffered IO, streams
- Let's work directly with the OS file handle!

Reality Check: Granularity (Cont'd)

- Experiment 3: `write()`

```
> echo 'main(){write(1, "Hello World\n", 13);}' | gcc -xc - -Os -w -static
-o hello3
> ./hello3
Hello World
> size hello3
   text    data     bss     dec     hex filename
 508138    1928    7052  517118    7e3fe hello3
```

512 KiB!

- 517703 compared to 517118 – a net saving of 585 bytes (0.1%) **:-(**
- Experiment 4: empty program

```
> echo 'main(){}' | gcc -xc - -Os -w -static -o hello4
> size hello4
   text    data     bss     dec     hex filename
 508074    1928    7052  517054    7e3be hello4
```

Hm...

- `objdump -D --reloc hello4 | grep printf | wc -l` yields still **2611** matches!
- It's the startup code!

Reality Check: Granularity (Cont'd)

- Experiment 5: `write()`, no startup code

```
> echo '_start(){write(1, "Hello World\n", 13);_exit(0);}' | gcc -xc - -Os -w
-static -nostartfiles -o hello5
> size hello5
   text    data     bss     dec     hex filename
    597       0       4     601     259 hello5
> ./hello5
Segmentation fault
```

0.5 KiB :-|

but segfault :-(

- Even a simple `write()` cannot be issued without the complete initialization.
- Last ressort: invoke the syscall directly!
- Experiment 6: `SYS_write()`

```
> echo '_start(){syscall(4, 1, "Hello World\n", 13);_exit(0);}' | gcc -xc - -Os
-w -static -nostartfiles -o hello6
> size hello6
   text    data     bss     dec     hex filename
    293       0       4     297     129 hello6
> ./hello6
Hello World
```

0.25 KiB :-)

Reality Check: Lessons Learned

> **297 ⟷ 517703 Bytes!**
>
> On Linux/glibc, a simple "Hello World" application takes **1750 times** more memory than necessary!

- However, is this a problem?
  - The glibc has been designed for a "standard case"
    - Large, multithreaded, IO-intensive UNIX application
    - Assumption: *every* program uses `malloc()`, `printf()`, ...
  - Variability has been traded for Granularity

> **Every Program?**
>
> *"I know of no feature that is always needed. When we say that two functions are almost always used together, we should remember that "almost" is a euphemism for "not"."*
>
> Parnas 1979: *"Designing Software for Ease of Extension and Contraction"* [8]

Reality Check: Lessons Learned

297 ⟷ 517703 Bytes!

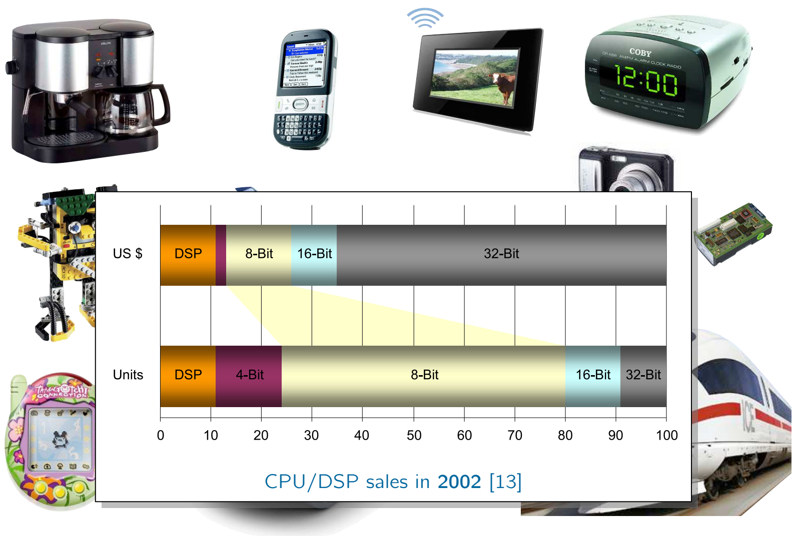
On Linux/glibc, a simple “Hello World” application takes **1750 times** more memory than necessary!

- However, is this a problem?
  - The glibc has been designed for a “standard case”
    - Large, multithreaded, IO-intensive UNIX application
    - Assumption: *every* program uses `malloc()`, `printf()`, ...
  - Variability has been traded for Granularity
- Assumption: The GP operating system will compensate for it...
  - Virtual memory ⇝ memory is not an issue (but is that a reason to waste it?)
  - Shared libraries ⇝ memory is actually shared between processes (unless we relocate the symbols, e.g., for address-space randomization...)

What about other domains?

Agenda

A Different Domain: Embedded Systems



CPU/DSP sales in **2002** [13]

The ATmega μC Family (8-Bit)

Type	Flash	SRAM	IO	Timer 8/16	UART	SPI	ADC	PWM	EUR
ATTINY13	1 KiB	64 B	6	1/-	-	-	1*4	-	0.86
ATTINY2313	2 KiB	128 B	18	1/1	-	1	-	-	0.99
ATMEGA48	4 KiB	512 B	23	2/1	1	1	8*10	6	1.40
ATMEGA16	16 KiB	1024 B	32	2/1	1	1	8*10	4	2.05
ATMEGA32	32 KiB	2048 B	32	2/1	1	1	8*10	4	3.65
ATMEGA64	64 KiB	4096 B	53	2/2	2	1	8*10	8	5.70
ATMEGA128	128 KiB	4096 B	53	2/2	2	1	8*10	8	7.35
ATMEGA256	256 KiB	8192 B	86	2/2	4	1	16*10	16	8.99

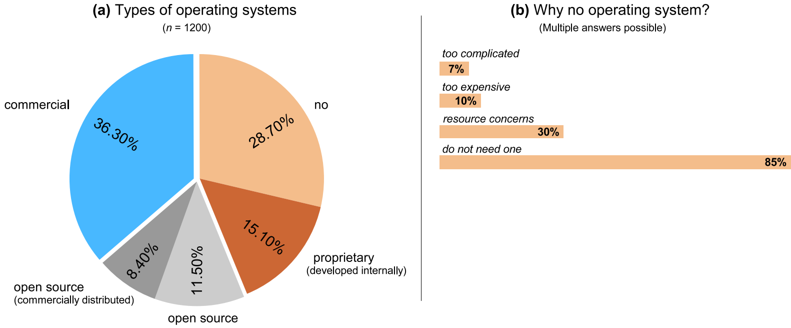
Prices and features of ATmega variants (excerpt, Reichelt April 2015)

Limited Resources

- Flash is limited, RAM is extremly limited
- A **few bytes** can have a **massive impact on** per-unit **costs**

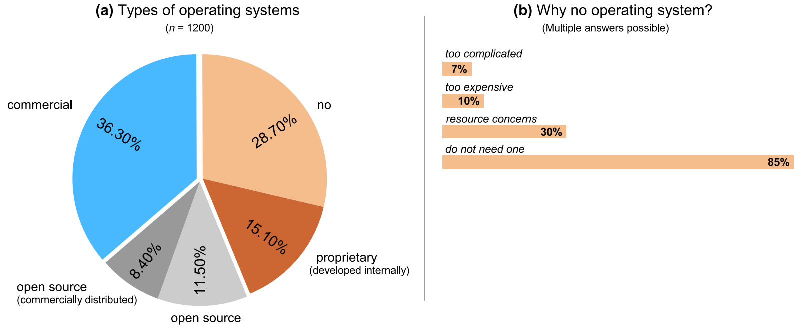
⇝ The “glibc approach” is **doomed to fail**!

The Role of the Operating System



Operating systems (not) employed in embedded-system projects in 2006 [12]

The Role of the Operating System



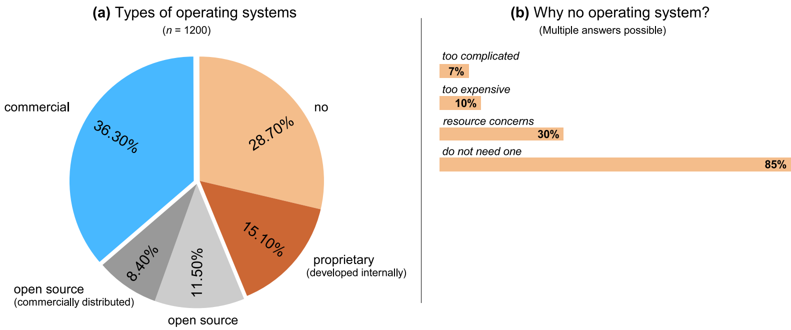
Operating systems (not) employed in embedded-system projects in 2006 [12]

> 40% of all projects use "in house" OS functionality!

Wide-spread fear of the resource overhead of GP operating systems

- OS functionality is developed "side-by-side" with the applications
- This leads to very high "hidden" development costs [14]

The Role of the Operating System



Operating systems (not) employed in embedded-system projects in 2006 [12]

Rest spreads over hundreds of different operating systems!

. . . , C{51, 166, 251}, CiAO, CMX RTOS, Contiki, C-Smart/Raven, eCos, eRTOS, Embos, Ercos, Euros Plus, FreeRTOS, Hi Ross, Hynet-OS, LynxOS, MicroX/OS-II, Nucleus, OS-9, OSE, OSEK {Flex, Turbo, Plus}, OSEKtime, Precise/MQX, Precise/RTCS, proOSEK, pSOS, PURE, PXROS, QNX, Realos, RTMOSxx, Real Time Architect, RTA, RTX{51, 166, 251}, RTXC, Softune, SSXS RTOS, ThreadX, TinyOS, Tresos, VRTX, VxWorks, . . .

⇝ The "glibc approach" (one size fits all) **does not work**!

Between a Rock and a Hard Place...

functional and nonfunctional *requirements*

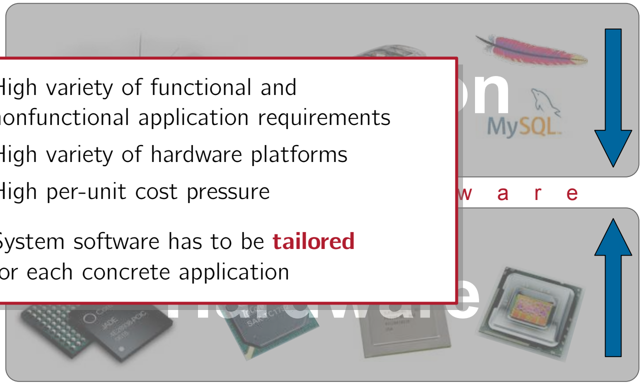
tasks
sockets
file system
...
event latency
safety
...

ISA
IRQ handling
MMU / MPU
...
cache size
coherence
IRQ latency
...

functional and nonfunctional *properties*

Between a Rock and a Hard Place...

functional and nonfunctional *requirements*



tasks
sockets
file system
...
event latency
safety
...

ISA
IRQ handling
MMU / MPU
...
cache size
coherence
IRQ latency
...

- High variety of functional and nonfunctional application requirements
- High variety of hardware platforms
- High per-unit cost pressure

⇝ System software has to be **tailored** for each concrete application

functional and nonfunctional *properties*

Customizing / Tailoring

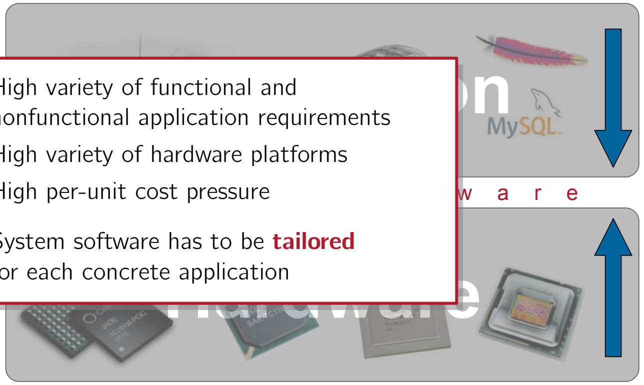
Customizing/Tailoring (Definition 3)

Customizing or tailoring is the activity of modifying existing system software in order to fulfill the requirements of some particular application.

This calls for **granularity** and **variability**!

Between a Rock and a Hard Place...

functional and nonfunctional *requirements*



tasks
sockets
file system
...
event latency
safety
...

ISA
IRQ handling
MMU / MPU
...
cache size
coherence
IRQ latency
...

- High variety of functional and nonfunctional application requirements
- High variety of hardware platforms
- High per-unit cost pressure

⇝ System software has to be **tailored** for each concrete application

functional and nonfunctional *properties*

Agenda

What to do?

297 ⟷ 517703 Bytes! **Why?**

On Linux/glibc, a simple "Hello World" application takes **1750 times** more memory than necessary!

- Reason: software structure
  - Trade-off between reuse ⟺ coupling
    ⇝ by extensive internal reuse, glibc has become an all-or-nothing blob
- Reason: software interface
  - C standard defines `printf()` as a swiss army knife [3, §7.19.6]
    ⇝ `printf()` has become a "god method" [1]
- Reason: language and tool chain
  - Compiler/linker work on the granularity of symbols or even object files
    ⇝ dead code is not effectively eliminated

What to do?

297 ⟷ 517703 Bytes! **Why?**

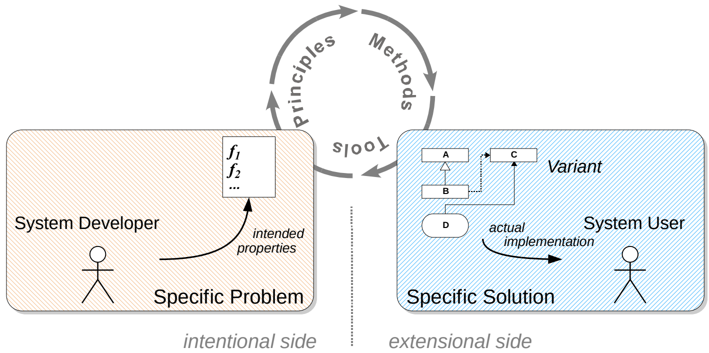
On Linux/glibc, a simple "Hello World" application takes **1750 times** more memory than necessary!

⇝ **Konfigurierbare Systemsoftware – KSS**

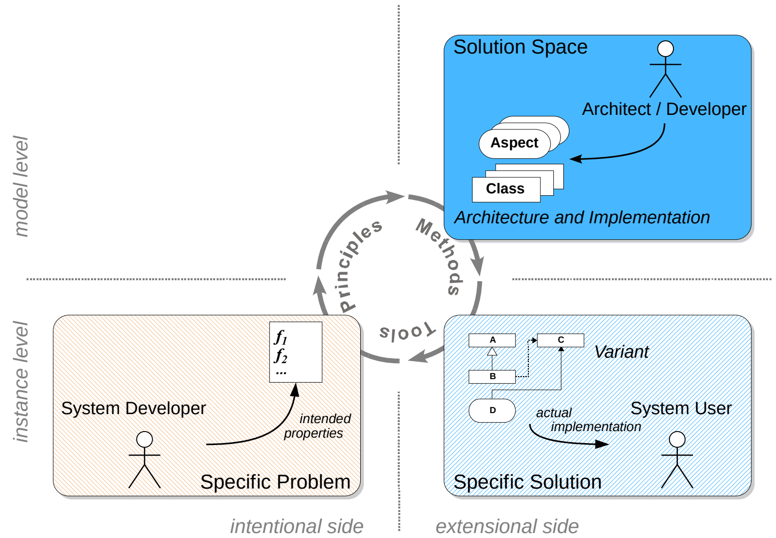
Througout the software development cycle, **variability** and **granularity** have to be considered as primary design goals from the very beginning!

In KSS you will learn about **principles**, **methods**, and **tools** to achieve this.

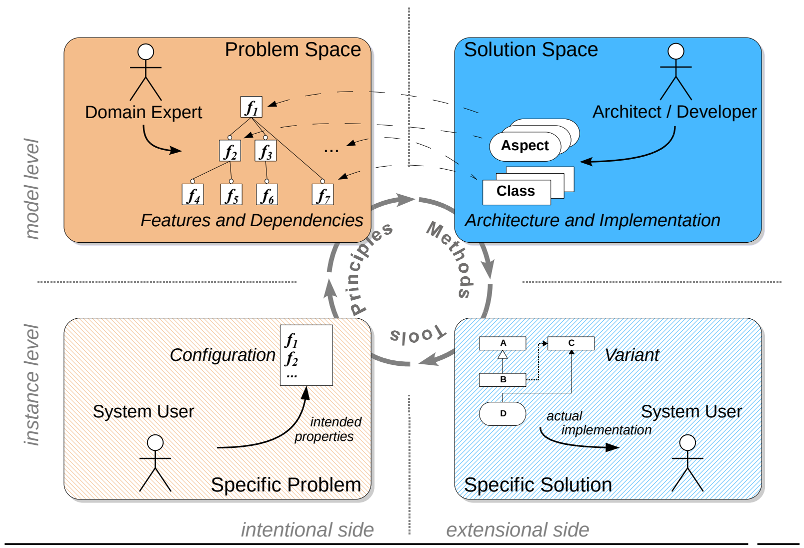
Individually Developed Software Product



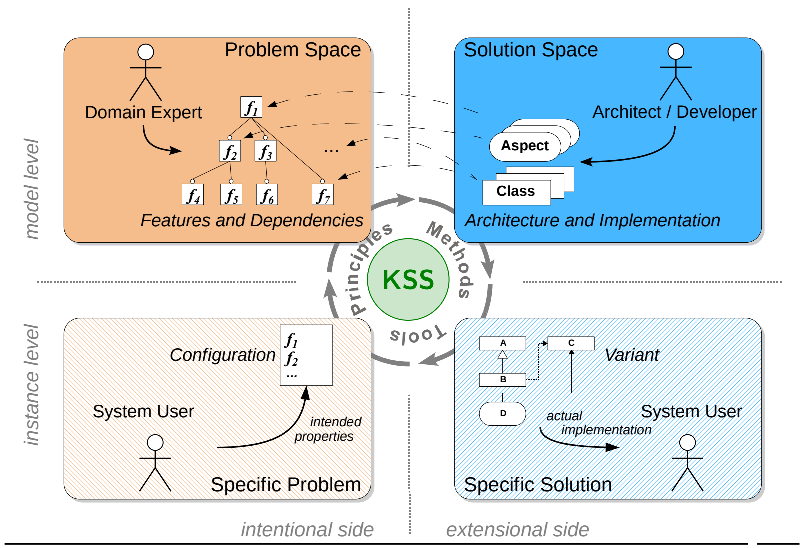
Software Product Derived from Reusable Assets



Configurable Software – Software Product Line



Configurable Software – Software Product Line



Agenda

Learning Objectives

- **Improve** your understanding of the design and development of low-level system software
  - Starting point: "Betriebssysteme" [BS]
  - Focus: Static configuration and tailoring
- **Expand** your knowledge by new software engineering methods and language techniques for configurable system software
  - Software families and software product lines [7]
  - Aspect-oriented and generative programming in C/C++ [10]
- **Apply** these techniques in the context of current operating-system research projects
  - CiAO, SLOTH, VAMOS, DanceOS [2, 5, 9, 11]
  - Get prepared for a master thesis or project in the field!

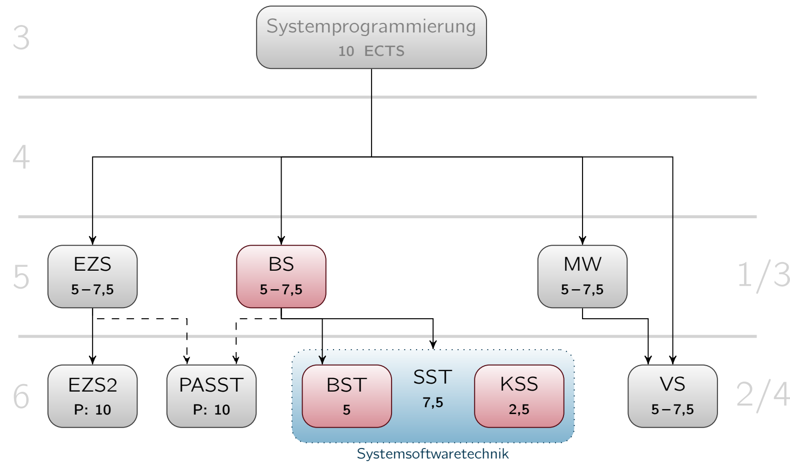
Topics

KSS ↦ Konfigurierbare Systemsoftware

- Software engineering
  - Variability
  - Product Lines
  - Aspects
  - Generators
- Operating systems
  - eCos [6]
  - CiAO [5]
  - Sloth [2]
  - Linux [11]

KSS – Einordnung

(Bachelor/Master)



Organisation: Systemsoftwaretechnik (SST)

- Modul Systemsoftwaretechnik (SST) **7.5 ECTS**
  1. Vorlesung Betriebssystemtechnik (BST) 2.5
     - Mo 14 – 16
     - 12–14 Vorlesungstermine
  2. Übungen zu Betriebssystemtechnik (BST-Ü) 2.5
     - Mi 10 – 12
     - 10 Übungstermine
  3. Vorlesung und Übung **Konfigurierbare Systemsoftware (KSS)** 2.5
     - Mo 12 – 14
     - 7 Vorlesungstermine, 1–2 Übungsaufgaben
     - Übung integriert in BST-Übung / Rechnerübung

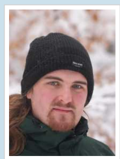
- ⇝ KSS kann **nur zusammen mit BST** belegt werden!
  - Es gibt keine 2.5 ECTS Module...
  - Wenn Bedarf besteht, wird KSS auf 5 ECTS erweitert

Organisation: Beteiligte

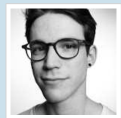
Vorlesung

Daniel Lohmann

Übung



Christian Dietrich



Valentin Rothberg

Semesterplanung

KW	Mo	Di	Mi	Do	Fr	Themen
16	13.04	14.04 BST VL1 (keine Übung)	15.04	16.04	17.04 KSS VL1	BST VL1: *Organisation und Einleitung* KSS VL1: *Introduction, Motivation and Concept*
17	20.04	21.04 BST VL2 TÜ BST A1	22.04	23.04	24.04	BST VL2: *Systemaufruf*
18	27.04 KSS VL2	28.04 BST VL3 KSS VL3	29.04	30.04	01.05 *1. Mai*	KSS VL2: *Software Families and Software Product Lines* BST VL3: *Aspect-Oriented Programming, AspectC++* KSS VL3: *Aspect-Oriented Programming, AspectC++*
19	04.05 KSS VL4	05.05 BST VL4 TÜ KSS A1	06.05 Abgabe BST A1	07.05	08.05	KSS VL4: *Aspect-Aware Design, CiAO* BST VL4: *Hierarchien*
20	11.05 KSS VL5	12.05 BST VL5 KSS VL6	13.05	14.05 *Himmelf.*	15.05	KSS VL5: *Variability in the Large, VAMOS* BST VL5: *Generative Programming, Sloth* KSS VL6: *Generative Programming, Sloth*
21	18.05 KSS VL7	19.05 BST VL6 TÜ BST A2	20.05	21.05 *Anstich*	22.05	KSS VL7: *Conclusion, Summary* BST VL6: *Adressraummodelle*
22	25.05 *Pfingsten/Berg*	26.05	27.05	28.05	29.05	
23	01.06	02.06 BST VL7 (Rechnerübung)	03.06	04.06	05.06	BST VL7: *Sprachbasierung*
24	08.06	09.06 BST VL8 TÜ BST A3	10.06 Abgabe BST A2	11.06	12.06	BST VL8: *Interprozesskommunikation*
25	15.06	16.06 BST VL9	17.06	18.06	19.06	BST VL9: *Kommunikationsabstraktionen*
26	22.06	23.06 BST VL10	24.06	25.06	26.06	BST VL10: *Mitbenutzung*
27	29.06	30.06 BST VL11	01.07	02.07	03.07	BST VL11: *Bindelader*
28	06.07	07.07 BST VL12	08.07	09.07	10.07	BST VL12: *Nachlese*
29	13.07	14.07	15.07 Abgabe BST A3	16.07	17.07	

http://www4.informatik.uni-erlangen.de/Lehre/SS16/V_KSS

Referenzen

[1] Martin Fowler and Kendall Scott. *UML konzentriert*. 2nd ed. Addison-Wesley, 2000. ISBN: 3-8273-1617-0.

[2] Wanja Hofer, Daniel Lohmann, Fabian Scheler, and Wolfgang Schröder-Preikschat. "Sloth: Threads as Interrupts". In: *Proceedings of the 30th IEEE International Symposium on Real-Time Systems (RTSS '09)*. (Washington, D.C., USA, Dec. 1–4, 2009). IEEE Computer Society Press, Dec. 2009, pp. 204–213. ISBN: 978-0-7695-3875-4. DOI: `10.1109/RTSS.2009.18`.

[3] International Organization for Standardization. *ISO/IEC 9899:TC2: Programming languages — C*. Geneva, Switzerland: International Organization for Standardization, 2005. URL: `http://www.open-std.org/JTC1/SC22/wg14/www/docs/n1124.pdf`.

[4] A.M. Lister and R.D. Eager. *Fundamentals of Operating Systems*. 5th. Macmillian, 1993. ISBN: 0-333-46986-0.

[BS] Daniel Lohmann. *Betriebssysteme*. Vorlesung mit Übung. Friedrich-Alexander-Universität Erlangen-Nürnberg, Lehrstuhl für Informatik 4, 2015 (jährlich). URL: `https://www4.cs.fau.de/Lehre/WS15/V_BS`.

Referenzen (Cont'd)

[5] Daniel Lohmann, Wanja Hofer, Wolfgang Schröder-Preikschat, Jochen Streicher, and Olaf Spinczyk. "CiAO: An Aspect-Oriented Operating-System Family for Resource-Constrained Embedded Systems". In: *Proceedings of the 2009 USENIX Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, June 2009, pp. 215–228. ISBN: 978-1-931971-68-3.

[6] Daniel Lohmann, Fabian Scheler, Reinhard Tartler, Olaf Spinczyk, and Wolfgang Schröder-Preikschat. "A Quantitative Analysis of Aspects in the eCos Kernel". In: *Proceedings of the ACM SIGOPS/EuroSys European Conference on Computer Systems 2006 (EuroSys '06)*. (Leuven, Belgium). Ed. by Yolande Berbers and Willy Zwaenepoel. New York, NY, USA: ACM Press, Apr. 2006, pp. 191–204. ISBN: 1-59593-322-0. DOI: `10.1145/1218063.1217954`.

[7] Daniel Lohmann, Olaf Spinczyk, and Wolfgang Schröder-Preikschat. "Lean and Efficient System Software Product Lines: Where Aspects Beat Objects". In: *Transactions on AOSD II*. Ed. by Awais Rashid and Mehmet Aksit. Lecture Notes in Computer Science 4242. Springer-Verlag, 2006, pp. 227–255. DOI: `10.1007/11922827_8`.

[8] David Lorge Parnas. "Designing Software for Ease of Extension and Contraction". In: *IEEE Transactions on Software Engineering* SE-5.2 (1979), pp. 128–138.

Referenzen (Cont'd)

[9] Horst Schirmeier, Rüdiger Kapitza, Daniel Lohmann, and Olaf Spinczyk. "DanceOS: Towards Dependability Aspects in Configurable Embedded Operating Systems". In: *Proceedings of the 3rd HiPEAC Workshop on Design for Reliability (DFR '11)*. Ed. by Alex Orailoglu. Heraklion, Greece, Jan. 2011, pp. 21–26.

[10] Olaf Spinczyk and Daniel Lohmann. "The Design and Implementation of AspectC++". In: *Knowledge-Based Systems, Special Issue on Techniques to Produce Intelligent Secure Software* 20.7 (2007), pp. 636–651. DOI: `10.1016/j.knosys.2007.05.004`.

[11] Reinhard Tartler, Daniel Lohmann, Julio Sincero, and Wolfgang Schröder-Preikschat. "Feature Consistency in Compile-Time-Configurable System Software: Facing the Linux 10,000 Feature Problem". In: *Proceedings of the ACM SIGOPS/EuroSys European Conference on Computer Systems 2011 (EuroSys '11)*. (Salzburg, Austria). Ed. by Christoph M. Kirsch and Gernot Heiser. New York, NY, USA: ACM Press, Apr. 2011, pp. 47–60. ISBN: 978-1-4503-0634-8. DOI: `10.1145/1966445.1966451`.

[12] Jim Turley. "Operating Systems on the Rise". In: *embedded.com* (June 2006). `http://www.eetimes.com/author.asp?section_id=36&doc_id=1287524`. URL: `http://www.eetimes.com/author.asp?section_id=36\&doc_id=1287524`.

[13] Jim Turley. "The Two Percent Solution". In: *embedded.com* (Dec. 2002). `http://www.embedded.com/story/OEG20021217S0039`, visited 2011-04-08.

Referenzen (Cont'd)

[14] Collin Walls. *The Perfect RTOS*. Keynote at embedded world '04, Nuremberg, Germany. 2004.