Konfigurierbare Systemsoftware (KSS)

VL 1 – Einführung

Daniel Lohmann

Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität
Erlangen-Nürnberg

SS 14 – 2014-04-08

http://www4.informatik.uni-erlangen.de/Lehre/SS14/V_KSS
Agenda

1.1 Commodity Operating Systems Today
1.2 Reality Check: Granularity
1.3 The Domain of Embedded Systems
1.4 About KSS
1.5 KSS — Organization
1.6 References
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
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.

Big is beautiful?

“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]
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 $\leftrightarrow$ GP library $\leftrightarrow$ GP function)

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

1.1 Commodity Operating Systems Today
1.2 Reality Check: Granularity
1.3 The Domain of Embedded Systems
1.4 About KSS
1.5 KSS — Organization
1.6 References
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

Maybe the general-purpose printf() is just too powerful?
- supports many data types, formatting rules, ...
- implementation requires a complex parser for the format string

Let’s try the much more specialized puts()!
Experiment 2: **puts()**

```bash
> echo 'main(){puts("Hello World");}'; | gcc -xc -Os -w -static -o hello2
> ./hello2
Hello World
> size hello2
```

<table>
<thead>
<tr>
<th>text</th>
<th>data</th>
<th>bss</th>
<th>dec</th>
<th>hex</th>
<th>filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>508723</td>
<td>1928</td>
<td>7052</td>
<td><strong>517703</strong></td>
<td>7e647</td>
<td>hello2</td>
</tr>
</tbody>
</table>

That didn’t help much!

Maybe **puts()** is yet too powerful?

- buffered IO, streams

Let’s work directly with the OS file handle!
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
```

- 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
```

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

- 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
```

0.5 KiB :-|
but segfault :-(

0.25 KiB :-)
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

---

"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]
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?
1.1 Commodity Operating Systems Today
1.2 Reality Check: Granularity
1.3 The Domain of Embedded Systems
1.4 About KSS
1.5 KSS — Organization
1.6 References
A Different Domain: Embedded Systems

CPU/DSP sales in 2002 [13]
The ATmega µC Family (8-Bit)

<table>
<thead>
<tr>
<th>Type</th>
<th>Flash</th>
<th>SRAM</th>
<th>IO</th>
<th>Timer 8/16</th>
<th>UART</th>
<th>I²C</th>
<th>AD</th>
<th>Price (€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTINY11</td>
<td>1 KiB</td>
<td>6</td>
<td>6</td>
<td>1/-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.31</td>
</tr>
<tr>
<td>ATTINY13</td>
<td>1 KiB</td>
<td>64 B</td>
<td>6</td>
<td>1/-</td>
<td>-</td>
<td>-</td>
<td>4*10</td>
<td>0.66</td>
</tr>
<tr>
<td>ATTINY2313</td>
<td>2 KiB</td>
<td>128 B</td>
<td>18</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>1.06</td>
</tr>
<tr>
<td>ATMega4820</td>
<td>4 KiB</td>
<td>512 B</td>
<td>23</td>
<td>2/1</td>
<td>2</td>
<td>1</td>
<td>6*10</td>
<td>1.26</td>
</tr>
<tr>
<td>ATMega8515</td>
<td>8 KiB</td>
<td>512 B</td>
<td>35</td>
<td>1/1</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>2.04</td>
</tr>
<tr>
<td>ATMega8535</td>
<td>8 KiB</td>
<td>512 B</td>
<td>32</td>
<td>2/1</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>2.67</td>
</tr>
<tr>
<td>ATMega169</td>
<td>16 KiB</td>
<td>1024 B</td>
<td>54</td>
<td>2/1</td>
<td>1</td>
<td>1</td>
<td>8*10</td>
<td>4.03</td>
</tr>
<tr>
<td>ATMega64</td>
<td>64 KiB</td>
<td>4096 B</td>
<td>53</td>
<td>2/2</td>
<td>2</td>
<td>1</td>
<td>8*10</td>
<td>5.60</td>
</tr>
<tr>
<td>ATMega128</td>
<td>128 KiB</td>
<td>4096 B</td>
<td>53</td>
<td>2/2</td>
<td>2</td>
<td>1</td>
<td>8*10</td>
<td>7.91</td>
</tr>
</tbody>
</table>

Bulk prices and features of ATmega variants (excerpt, DigiKey 2006)

Limited Resources

- Flash is limited, RAM is extremely 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

(a) Types of operating systems (n = 1200)

- commercial: 36.30%
- open source (commercially distributed): 8.40%
- proprietary (developed internally): 15.10%
- no: 28.70%
- open source: 11.50%

(b) Why no operating system? (Multiple answers possible)
- too complicated: 7%
- too expensive: 10%
- resource concerns: 30%
- do not need one: 85%

Operating systems (not) employed in embedded-system projects in 2006 [12]
The Role of the Operating System

> 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

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]

- Commercial
  - 36.30%
- Proprietary (developed internally)
  - 15.10%
- Open source (commercially distributed)
  - 8.40%
- Open source
  - 11.50%
- No
  - 28.70%

(b) Why no operating system?

- Too complicated: 7%
- Too expensive: 10%
- Resource concerns: 30%
- Do not need one: 85%

The “glibc approach” (one size fits all) does not work!
Between a Rock and a Hard Place...

**functional and nonfunctional requirements**

**Application**

**System Software**

**Functional and nonfunctional properties**

**Hardware**

- tasks
- sockets
- file system
- event latency
- safety
- ISA
- IRQ handling
- MMU / MPU
- cache size
- coherence
- IRQ latency

© dl KSS (VL 1 | SS 14) 1 Motivation | 1.3 The Domain of Embedded Systems 1–16
Between a Rock and a Hard Place...

- 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 requirements**

**tasks**
- sockets
- file system

...event latency
...safety

**ISA**
- IRQ handling
- MMU / MPU

...cache size
...coherence
...IRQ latency
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**

- 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**

tasks sockets file system

... event latency safety ...

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

© dl KSS (VL 1 | SS 14) 1 Motivation | 1.3 The Domain of Embedded Systems
Agenda

1.1 Commodity Operating Systems Today
1.2 Reality Check: Granularity
1.3 The Domain of Embedded Systems
1.4 About KSS
1.5 KSS — Organization
1.6 References
What to do?

297 $\leftrightarrow$ 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 $\iff$ coupling
  $\leadsto$ by extensive internal reuse, glibc has become an all-or-nothing blob

Reason: software interface

- C standard defines `printf()` as a swiss army knife $\iff$ `printf()` has become a “god method”

Reason: language and tool chain

- Compiler/linker work on the granularity of symbols or even object files
  $\leadsto$ dead code is not effectively eliminated
What to do?

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

Konfigurierbare Systemsoftware – KSS

Throughout 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

System Developer

Specific Problem

$\begin{align*}
f_1 \\
f_2 \\
\vdots
\end{align*}$

intended properties

Specific Solution

Variant

System User

intentional side

extensional side

principles

Methods

Tools
Software Product Derived from Reusable Assets

Specific Problem

Architect / Developer

Solution Space

Class

Aspect

System Developer

Variant

System User

Architecture and Implementation

intentional side

extensional side

model level

instance level

© dl KSS (VL 1 | SS 14) 1 Motivation | 1.4 About KSS
Configurable Software – Software Product Line

Problem Space
- Domain Expert
- Features and Dependencies
  - $f_1$
  - $f_2$
  - $f_3$
  - $f_4$
  - $f_5$
  - $f_6$
  - $f_7$

Solution Space
- Architect / Developer
- Architecture and Implementation
  - Aspect
  - Class

Specific Problem
- System User
- Specific Solution
- Variant
- System User

Features and Dependencies

Unintended side

intentional side    extensional side

© dl KSS (VL 1 | SS 14) 1 Motivation | 1.4 About KSS
1.1 Commodity Operating Systems Today
1.2 Reality Check: Granularity
1.3 The Domain of Embedded Systems
1.4 About KSS

1.5 KSS — Organization
   - Objectives
   - Einordnung
   - Semesterplanung

1.6 References
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 \rightarrow

Konfigurierbare Systemsoftware

{ Software engineering

{ Variability

{ Product Lines

{ Aspects

{ Generators
eCos [6]

{ CiAO [5]

{ Sloth [2]

{ Linux [11]

Operating systems
Organisation: Systemsoftwaretechnik (SST)

Modul Systemsoftwaretechnik (SST) 7.5 ECTS

1. Vorlesung Betriebssystemtechnik (BST) 2.5
   - Mo 12–14
   - 12–14 Vorlesungstermine

2. Übungen zu Betriebssystemtechnik (BST-Ü) 2.5
   - Di 10–12
   - 12–14 Übungstermine/Rechnerübungen

3. Vorlesung und Übung Konfigurierbare Systemsoftware (KSS) 2.5
   - Do 14–16 (Vorlesung)
   - 7 Vorlesungstermine, 1 Übungsaufgabe, 1 Projekt
   - 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

Daniel Danner
Gabor Drescher

Projekt

Daniel Danner
Martin Hoffmann
Jens Schedel
?
<table>
<thead>
<tr>
<th>KW</th>
<th>Mo</th>
<th>Di</th>
<th>Mi</th>
<th>Do</th>
<th>Fr</th>
<th>Themen</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>07.04.</td>
<td>08.04.</td>
<td>09.04.</td>
<td>10.04.</td>
<td>11.04.</td>
<td>BST VL1: Organisation und Einleitung</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>KSS VL1: Introduction, Motivation and Concept</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>KSS VL2: Software Families and Software Product Lines</td>
</tr>
<tr>
<td>16</td>
<td>14.04.</td>
<td>15.04.</td>
<td>16.04.</td>
<td>17.04.</td>
<td>18.04.</td>
<td>BST VL2: Systemaufruf</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>KSS VL3: Aspect-Oriented Programming, AspectC++</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BST VL3: Betriebssystemarchitektur</td>
</tr>
<tr>
<td>17</td>
<td>21.04.</td>
<td>22.04.</td>
<td>23.04.</td>
<td>24.04.</td>
<td>25.04.</td>
<td>BST VL4: Betriebssystemarchitektur</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>KSS VL4: Aspect-Aware Design, CiAO</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BST VL5: Adressraumverwaltung</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>KSS VL5: Variability in the Large, VAMOS</td>
</tr>
<tr>
<td>18</td>
<td>28.04.</td>
<td>29.04.</td>
<td>30.04.</td>
<td>01.05.</td>
<td>02.05.</td>
<td>BST VL5: Adressraumverwaltung</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>KSS VL5: Variability in the Large, VAMOS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BST VL6: Adressraummodelle</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>KSS VL6: Generative Programming, Sloth</td>
</tr>
<tr>
<td>19</td>
<td>05.05.</td>
<td>06.05.</td>
<td>07.05.</td>
<td>08.05.</td>
<td>09.05.</td>
<td>BST VL6: Adressraummodelle</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>KSS VL6: Generative Programming, Sloth</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BST VL7: Sprachbasierung</td>
</tr>
<tr>
<td>20</td>
<td>12.05.</td>
<td>13.05.</td>
<td>14.05.</td>
<td>15.05.</td>
<td>16.05.</td>
<td>BST VL7: Sprachbasierung</td>
</tr>
<tr>
<td></td>
<td>BST VL5</td>
<td>TÜ BST A2</td>
<td>Abgabe BST A1</td>
<td>KSS VL5</td>
<td></td>
<td>KSS VL7: Conclusion, Summary</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BST VL8: Interprozesskommunikation</td>
</tr>
<tr>
<td>21</td>
<td>19.05.</td>
<td>20.05.</td>
<td>21.05.</td>
<td>22.05.</td>
<td>23.05.</td>
<td>BST VL8: Interprozesskommunikation</td>
</tr>
<tr>
<td></td>
<td>BST VL6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>KSS VL7: Conclusion, Summary</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BST VL9: Kommunikationsabstraktionen</td>
</tr>
<tr>
<td>22</td>
<td>26.05.</td>
<td>27.05.</td>
<td>28.05.</td>
<td>29.05.</td>
<td>30.05.</td>
<td>BST VL9: Kommunikationsabstraktionen</td>
</tr>
<tr>
<td></td>
<td>BST VL7</td>
<td></td>
<td>Abgabe KSS A1</td>
<td>KSS VL7</td>
<td></td>
<td>BST VL10: Mitbenutzung</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BST VL10: Mitbenutzung</td>
</tr>
<tr>
<td>23</td>
<td>02.06.</td>
<td>03.06.</td>
<td>04.06.</td>
<td>05.06.</td>
<td>06.06.</td>
<td>BST VL10: Mitbenutzung</td>
</tr>
<tr>
<td></td>
<td>BST VL8</td>
<td></td>
<td></td>
<td>Anstich</td>
<td></td>
<td>KSS VL8: Interprozesskommunikation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BST VL9: Kommunikationsabstraktionen</td>
</tr>
<tr>
<td>24</td>
<td>09.06.</td>
<td>10.06.</td>
<td>11.06.</td>
<td>12.06.</td>
<td>13.06.</td>
<td>BST VL9: Kommunikationsabstraktionen</td>
</tr>
<tr>
<td></td>
<td>Pfingsten/Berg</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>KSS VL8: Interprozesskommunikation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BST VL11: Bindelader</td>
</tr>
<tr>
<td>25</td>
<td>16.06.</td>
<td>17.06.</td>
<td>18.06.</td>
<td>19.06.</td>
<td>20.06.</td>
<td>BST VL11: Bindelader</td>
</tr>
<tr>
<td></td>
<td>BST VL9</td>
<td></td>
<td>Abgabe BST A2</td>
<td>Fronleich.</td>
<td></td>
<td>BST VL12: Nachlese</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BST VL12: Nachlese</td>
</tr>
<tr>
<td>26</td>
<td>23.06.</td>
<td>24.06.</td>
<td>25.06.</td>
<td>26.06.</td>
<td>27.06.</td>
<td>BST VL11: Bindelader</td>
</tr>
<tr>
<td></td>
<td>BST VL10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BST VL12: Nachlese</td>
</tr>
<tr>
<td>27</td>
<td>30.06.</td>
<td>01.07.</td>
<td>02.07.</td>
<td>03.07.</td>
<td>04.07.</td>
<td>BST VL11: Bindelader</td>
</tr>
<tr>
<td></td>
<td>BST VL11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BST VL12: Nachlese</td>
</tr>
<tr>
<td>28</td>
<td>07.07.</td>
<td>08.07.</td>
<td>09.07.</td>
<td>10.07.</td>
<td>11.07.</td>
<td>BST VL12: Nachlese</td>
</tr>
</tbody>
</table>

http://www4.informatik.uni-erlangen.de/Lehre/SS14/V_KSS
Referenzen


Referenzen (Cont’d)


