Seminar AspektOS

DSLs für Systemsoftware:

DEVIL und BOSSA

Referent: Helmut Wünsch

Übersicht

- **DSLs: kurze Einführung**
- **DEVIL**
  - Motivation
  - Prinzip
  - Praxisbeispiel Busmousedriver
    - Implementierung konventionell
    - Implementierung Mittels DEVIL
    - Vergleich
  - Ausblick
- **BOSSA**
  - Motivation
  - Prinzip
  - Praxisbeispiel Linux-Scheduler
  - Virtuelle Scheduler
  - Kritische Betrachtung
- **Quellen**

Was sind DSLs?

- **D**omain **S**pecific **L**anguage
- Programmiersprache für ganz bestimmten Problembereich (↔ **G**eneral **P**urpose **L**anguage)
- In der Regel mehr deklarativ als imperativ
- Bekannte Beispiele aus der Unix-Welt:
  bash, make, postscript,...

DEVIL

Motivation

- Immer komplexere Hardware auf dem Markt
  → Treiberunterstützung
- Seit Jahren keine merklichen Innovationen auf dem Gebiet der Treiberentwicklung
- Folge: Produktion von meist unlesbaren, schwer zu debuggende Treibercodes.
- → Fehleranfälligkeit!

Gesucht: Ein Werkzeug, das:

- den Treiberprogrammierer bei seiner Arbeit unterstützt
- ein gewisses Maß an Typsicherheit auf Low-Level-Ebene ermöglicht
- den Treibercode lesbar und wartbar hält

Lösungsansatz: Devil

- **DEV**ice **I**nterface **L**anguage
- Programmierer beschreibt Interface der Hardware in DEVIL-spezifischer Sprache
- DEVIL-Compiler bastelt daraus ein C-Headerfile mit Inline-Fuktionen (“stubs”)
- Im eigentlichen C-Treibercode komfortabler und sicherer Zugriff auf Hardware mittels der generierten DEVIL-stubs

Bsp: Logitech Busmouse

Funktionale Schnittstelle der Busmouse Hardware:

- **X-Koordinate(relativ)**
- **Y-Koordinate(relativ)**
- **Buttons**

Busmouse(2)

Hardwareports:

Bereitstellung der Hardwarefunktionalität über vier Hardwareregister:

(BASE = Hardwareadresse der Busmousekarte, z.B. 0x23c)

- **Data Port** *(BASE + 0)*
- Signature Port *(BASE + 1)*
- **Control Port** *(BASE + 2)*
- Config Port *(BASE + 3)*

Busmouse(3)

Funktionalität Data / ControlPort:

Data Port (8-Bit-Register):

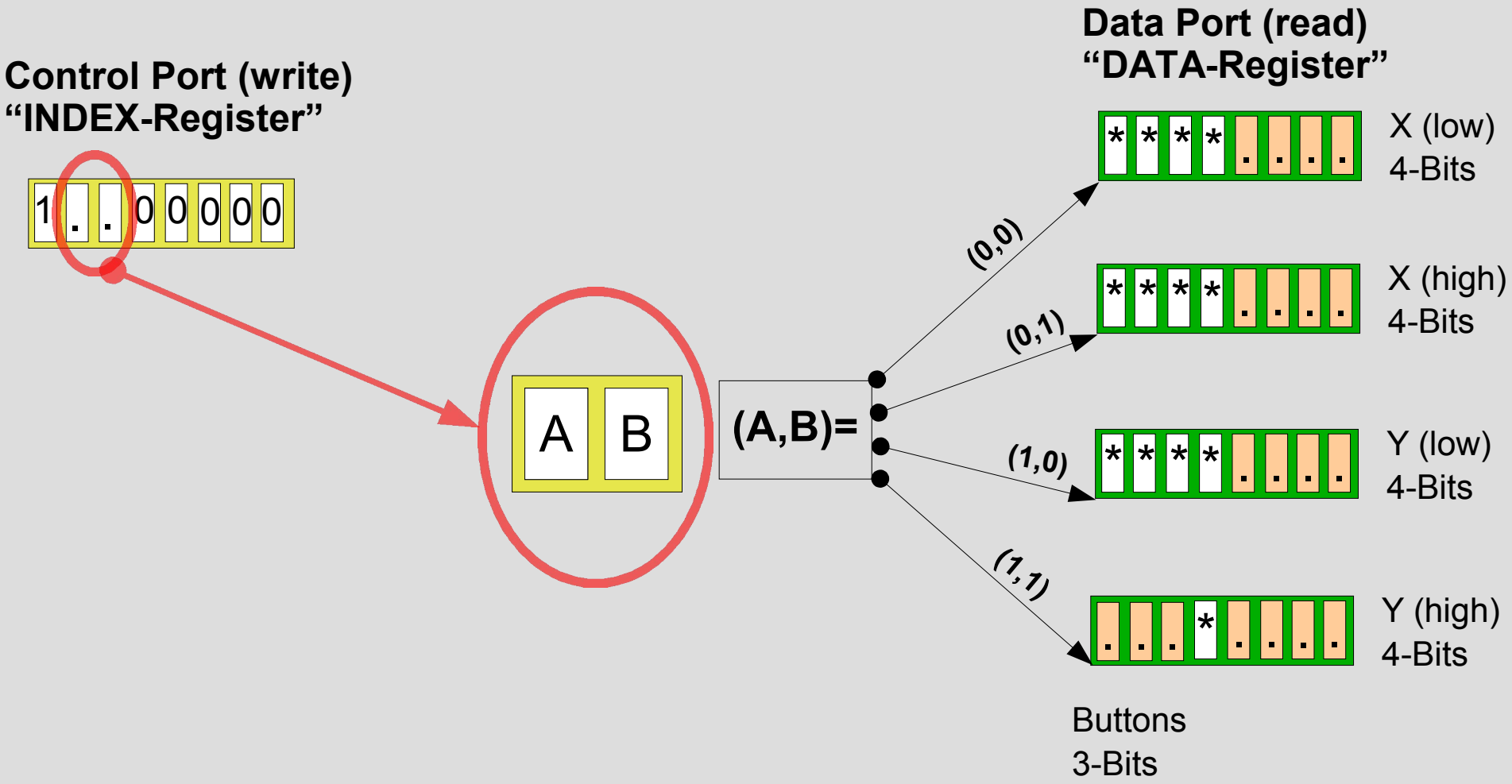
- X-Koordinate (low / high)
- Y-Koordinate (low / high)
- Buttons (Bitmap)

Control Port (8-Bit-Register):

- Auswahl der Funktion des Data-Ports

Busmouse(4)

Funktionsauswahl Data Port:



Konventionelle Umsetzung: Busmousedriver in Linux 2.6.5

~/drivers/input/mouse/logibm.c :

Deklarationen

```
#define LOGIBM_BASE              0x23c

[...]

#define LOGIBM_DATA_PORT         LOGIBM_BASE + 0
#define LOGIBM_SIGNATURE_PORT    LOGIBM_BASE + 1
#define LOGIBM_CONTROL_PORT      LOGIBM_BASE + 2
#define LOGIBM_CONFIG_PORT       LOGIBM_BASE + 3

[...]

#define LOGIBM_READ_X_LOW        0x80
#define LOGIBM_READ_X_HIGH       0xa0
#define LOGIBM_READ_Y_LOW        0xc0
#define LOGIBM_READ_Y_HIGH       0xe0
```

Konventionelle Umsetzung(2): Busmousedriver in Linux 2.6.5

/usr/src/linux/drivers/input/mouse/logibm.c :

Interrupthandler:

```
static irqreturn_t logibm_interrupt
(int irq, void *dev_id, struct pt_regs *regs){
        char dx, dy;
        unsigned char buttons;

        outb(LOGIBM_READ_X_LOW, LOGIBM_CONTROL_PORT);
        dx = (inb(LOGIBM_DATA_PORT) & 0xf);
        outb(LOGIBM_READ_X_HIGH, LOGIBM_CONTROL_PORT);
        dx |= (inb(LOGIBM_DATA_PORT) & 0xf) << 4;
        outb(LOGIBM_READ_Y_LOW, LOGIBM_CONTROL_PORT);
        dy = (inb(LOGIBM_DATA_PORT) & 0xf);
        outb(LOGIBM_READ_Y_HIGH, LOGIBM_CONTROL_PORT);
        buttons = inb(LOGIBM_DATA_PORT);
        dy |= (buttons & 0xf) << 4;
        buttons = ~buttons >> 5;
        outb(LOGIBM_ENABLE_IRQ, LOGIBM_CONTROL_PORT);

       [...]

        return IRQ_HANDLED;
}
```

Konventionelle Umsetzung(3): Busmousedriver in Linux 2.6.5

Quintessenz:

Grauenhafter Treibercode!!!

Neuer Versuch: Implementierung mittels DEVIL

Erstelle Datei in DEVIL-Code (z.B. “busmouse.dil”):

```
device logitech_busmouse (base : bit[8] port @ {0..3})
{









}
```

Gerätedefinition

Neuer Versuch: Implementierung mittels DEVIL

Erstelle Datei in DEVIL-Code (z.B. “busmouse.dil”):

```
device logitech_busmouse (base : bit[8] port @ {0..3})
{
  // index register
  register index_reg = write base @ 2, mask '1..00000' : bit[8];









}
```

Registerdefinition:

Indexregister auf Port BASE+2
(= Control Port)

write-only!

Neuer Versuch: Implementierung mittels DEVIL

Erstelle Datei in DEVIL-Code (z.B. “busmouse.dil”):

```
device logitech_busmouse (base : bit[8] port @ {0..3})
{
  // index register
  register index_reg = write base @ 2, mask '1..00000' : bit[8];
  private variable index = index_reg[6..5] : int(2);









}
```

Variablendefinition:

Bits 5-6 definieren
2-Bit INT-Variable

Neuer Versuch: Implementierung mittels DEVIL

Erstelle Datei in DEVIL-Code (z.B. “busmouse.dil”):

```
device logitech_busmouse (base : bit[8] port @ {0..3})
{
  // index register
  register index_reg = write base @ 2, mask '1..00000' : bit[8];
  private variable index = index_reg[6..5] : int(2);

  // Data register
  register x_low  = read base @ 0, pre {index = 0}, mask '****....' : bit[8];
  register x_high = read base @ 0, pre {index = 1}, mask '****....' : bit[8];
  register y_low  = read base @ 0, pre {index = 2}, mask '****....' : bit[8];
  register y_high = read base @ 0, pre {index = 3}, mask '...*....' : bit[8];









}
```

Registerdefinition mit Pre-Action:

Datenregister auf 4 Registervariablen “gemultiplext”

Neuer Versuch: Implementierung mittels DEVIL

Erstelle Datei in DEVIL-Code (z.B. “busmouse.dil”):

```
device logitech_busmouse (base : bit[8] port @ {0..3})
{
  // index register
  register index_reg = write base @ 2, mask '1..00000' : bit[8];
  private variable index = index_reg[6..5] : int(2);

  // Data register
  register x_low  = read base @ 0, pre {index = 0}, mask '****....' : bit[8];
  register x_high = read base @ 0, pre {index = 1}, mask '****....' : bit[8];
  register y_low  = read base @ 0, pre {index = 2}, mask '****....' : bit[8];
  register y_high = read base @ 0, pre {index = 3}, mask '...*....' : bit[8];

  structure mouste_state = {
     variable dx = x_high[3..0] # x_low[3..0], volatile : signed int(8);
     variable dy = y_high[3..0] # y_low[3..0], volatile : signed int(8);
     variable buttons = y_high[7..5], volatile : int(3);
  };

}
```

Variablendefinition (Typ volatile):

Zusammengesetzte “cache”-Werte für dx,dy,buttons

(exportierte Variablen)

Neuer Versuch: Implementierung mittels DEVIL

Erstelle Datei in DEVIL-Code (z.B. “busmouse.dil”):

```
device logitech_busmouse (base : bit[8] port @ {0..3})
{
  // index register
  register index_reg = write base @ 2, mask '1..00000' : bit[8];
  private variable index = index_reg[6..5] : int(2);

  // Data register
  register x_low  = read base @ 0, pre {index = 0}, mask '****....' : bit[8];
  register x_high = read base @ 0, pre {index = 1}, mask '****....' : bit[8];
  register y_low  = read base @ 0, pre {index = 2}, mask '****....' : bit[8];
  register y_high = read base @ 0, pre {index = 3}, mask '...*....' : bit[8];

  structure mouste_state = {
     variable dx = x_high[3..0] # x_low[3..0], volatile : signed int(8);
     variable dy = y_high[3..0] # y_low[3..0], volatile : signed int(8);
     variable buttons = y_high[7..5], volatile : int(3);
  };
  // signature register (sr)
  register sr = base @ 1 : bit[8];
  variable sig = sr, volatile, write trigger : int(8);

  // configuration register (CR)
  register cr = write base @ 3, mask '1001000.' : bit[8];
  variable config = cr[0] : {CONFIGURATION => '1', DEFAULT_MODE  => '0'};

  // interrupt register
  register interruption = write base @ 2, mask '000.0000' : bit[8];
  variable interrupt = interruption[4] : {ENABLE  => '0', DISABLE => '1'};
}
```

Weitere DEVIL-Konstrukte:

'trigger', Enumeration

Neuer Versuch: Implementierung mittels DEVIL

Erstelle Datei in DEVIL-Code (z.B. “busmouse.dil”):

```
device logitech_busmouse (base : bit[8] port @ {0..3})
{
  // index register
  register index_reg = write base @ 2, mask '1..00000' : bit[8];
  private variable index = index_reg[6..5] : int(2);

  // Data register
  register x_low  = read base @ 0, pre {index = 0}, mask '****....' : bit[8];
  register x_high = read base @ 0, pre {index = 1}, mask '****....' : bit[8];
  register y_low  = read base @ 0, pre {index = 2}, mask '****....' : bit[8];
  register y_high = read base @ 0, pre {index = 3}, mask '...*....' : bit[8];

  structure mouste_state = {
     variable dx = x_high[3..0] # x_low[3..0], volatile : signed int(8);
     variable dy = y_high[3..0] # y_low[3..0], volatile : signed int(8);
     variable buttons = y_high[7..5], volatile : int(3);
  };
  // signature register (sr)
  register sr = base @ 1 : bit[8];
  variable sig = sr, volatile, write trigger : int(8);

  // configuration register (CR)
  register cr = write base @ 3, mask '1001000.' : bit[8];
  variable config = cr[0] : {CONFIGURATION => '1', DEFAULT_MODE  => '0'};

  // interrupt register
  register interruption = write base @ 2, mask '000.0000' : bit[8];
  variable interrupt = interruption[4] : {ENABLE  => '0', DISABLE => '1'};
}
```

Komplette DEVIL-Spezifikation

Verwendung

Erstellte .dil-Datei durch den DEVIL-Compiler jagen:

```
# ./taz busmouse.dil
```

```
busmouse.dil.h
```

Verwendung

Erstellte .dil-Datei durch den DEVIL-Compiler jagen:

```
# ./taz busmouse.dil
```

busmouse.dil.h

```
[...]

#define __dil_dev_param__ dev_logitech_busmouse *dev
#define __dil_dev_param_comma__ dev_logitech_busmouse *dev,
#define __dil_func(name) name
#define __dil_access_cache(x) __dil_xcat((*dev)., x)

[...]

static inline void
__dil_func(reg_set_index_reg) (__dil_dev_param_comma__ u8 v)
{
  outb ((u8)v, __dil_access_cache (__dil_base__) + 2);
}

[...]

static inline void
__dil_func(get_mouste_state) (__dil_dev_param__)
{
  (__dil_access_cache (cache_mouste_state)).cache_get_x_high = __dil_func(reg_get_x_high)...
  (__dil_access_cache (cache_mouste_state)).cache_get_x_low = __dil_func(reg_get_x_low)...
  (__dil_access_cache (cache_mouste_state)).cache_get_y_high = __dil_func(reg_get_y_high)...
  (__dil_access_cache (cache_mouste_state)).cache_get_y_low = __dil_func(reg_get_y_low)...
}
```

Verwendung

Busmouse Linuxtreiber mit DEVIL-Stubs:

```
[...]

#include "busmouse.dil.h"

[...]

bm_init_logitech_busmouse (0x23c);

[...]

static irqreturn_t logibm_interrupt (int irq, void *dev_id, struct pt_regs *regs){
   char dx, dy;
   unsigned char buttons;

   bm_get_mouste_state ();

   dx = bm_get_dx ();
   dy = bm_get_dy ();
   buttons = bm_get_buttons ();

   [...]

}
```

Vergleich
DEVIL - Konventionell

```
[...]

#include "busmouse.dil.h"

[...]

static irqreturn_t logibm_interrupt
(int irq, void *dev_id, struct pt_regs *regs){

        char dx, dy;
        unsigned char buttons;

        bm_get_mouste_state ();
        dx = bm_get_dx ();
        dy = bm_get_dy ();
        buttons = bm_get_buttons ();

       [...]

}

[...]
```

```
[...]

#define LOGIBM_BASE             0x23c

[...]

#define LOGIBM_DATA_PORT        LOGIBM_BASE + 0
#define LOGIBM_SIGNATURE_PORT   LOGIBM_BASE + 1
#define LOGIBM_CONTROL_PORT     LOGIBM_BASE + 2
#define LOGIBM_CONFIG_PORT      LOGIBM_BASE + 3

[...]

#define LOGIBM_READ_X_LOW       0x80
#define LOGIBM_READ_X_HIGH      0xa0
#define LOGIBM_READ_Y_LOW       0xc0
#define LOGIBM_READ_Y_HIGH      0xe0


static irqreturn_t logibm_interrupt
(int irq, void *dev_id, struct pt_regs *regs){

        char dx, dy;
        unsigned char buttons;

        outb(LOGIBM_READ_X_LOW, LOGIBM_CONTROL_PORT);
        dx = (inb(LOGIBM_DATA_PORT) & 0xf);
        outb(LOGIBM_READ_X_HIGH, LOGIBM_CONTROL_PORT);
        dx |= (inb(LOGIBM_DATA_PORT) & 0xf) << 4;
        outb(LOGIBM_READ_Y_LOW, LOGIBM_CONTROL_PORT);
        dy = (inb(LOGIBM_DATA_PORT) & 0xf);
        outb(LOGIBM_READ_Y_HIGH, LOGIBM_CONTROL_PORT);
        buttons = inb(LOGIBM_DATA_PORT);
        dy |= (buttons & 0xf) << 4;
        buttons = ~buttons >> 5;
        outb(LOGIBM_ENABLE_IRQ, LOGIBM_CONTROL_PORT);

        [...]
}
```

DEVIL:
kritische Betrachtung

Pro:

- Sauberer Code
- Sicherheitsmechanismen
- Plattform/Architekturunabhängigkeit???

Kontra:

- Performanceeinbussen (um die 10%)
- DEVIL-Kompilerabhängigkeit

DEVIL: Ausblick

- Geschwindigkeitsoptimierung
- **Vision**: Komplette DEVIL-Definition der Hardware schon vom Hersteller bereitgestellt.
- Aufbau einer Public Domain Library von DevilSpecs

BOSSA

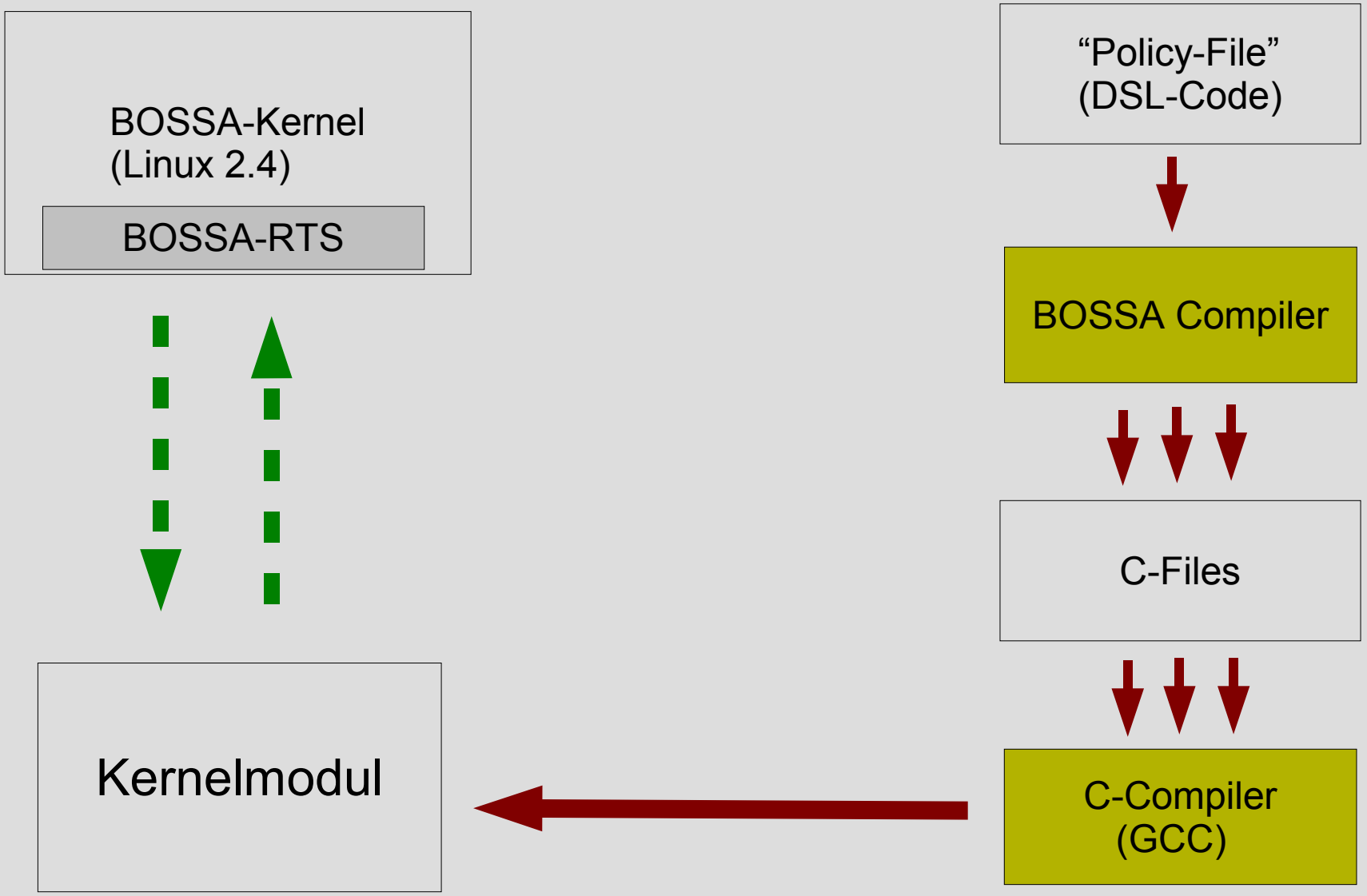
Motivation:

- Scheduler DAS zentrale Element eines jeden Multitasking-OS
- Scheduler-Code: Quer über das gesamte System verteilt
- “Schnell mal eine andere Scheduling-Strategie ausprobieren”
- “Wechsel” des Schedulers normalerweise nur für ausgemachte Kernelexperten möglich

Lösung:

- BOSSA
- Speziell gepatchter Linux-2.4-Kern
- Zentralisierung der gesamten Scheduleraspekte an definierten Punkt
- Entwurf des kompletten Schedulers in eventbasierter DSL → “**Policy-File**”
- Kompilierung → Eigener Scheduler als Kernelmodul ladbar

Prinzip:


BOSSA-Kernel
(Linux 2.4)
BOSSA-RTS
“Policy-File”
(DSL-Code)
BOSSA Compiler
C-Files
C-Compiler
(GCC)
Kernelmodul

BOSSA-DSL: Entwurf eines Policy-Files

- Eventbasierte, C-ähnliche Hochsprache
- Erfassung von allen Schedulerrelevanten Events
- Kaum Linux-spezifisches Backgroundwissen notwendig
- Fast jedes Schedulerverhalten beschreibbar

Policy-File: prinzipieller Aufbau:

```
default scheduler Linux = {
  .
  type policy_t = ...
  process = ...
  states = ...
  .
  .

  handler (event e) {
    .
    .
    .
  }

  interface = {
    void attach ...
    void detach ...
    .
    .
    .
  }

}
```

Deklarationsteil

Eventhandler

Interface-Fkt.

Bsp. Policy-File: Linux 2.4 Schedulerverhalten

```
default scheduler Linux = {
  type policy_t = enum {BOSSA_SCHED_FIFO, BOSSA_SCHED_RR, BOSSA_SCHED_OTHER};

  process = {
    policy_t policy;
    int      rt_priority;
    int      priority;
    int      ticks;
    system struct mm_struct system active_mm; // process virtual address space
  }

  system struct mm_struct running_active_mm = NULL;

  states = {
    RUNNING running : process;
    READY ready : sorted queue;
    READY expired : queue;
    READY yield : process;
    BLOCKED blocked : queue;
    TERMINATED terminated;
  }

  ordering_criteria = {
    highest rt_priority, highest (true ? (priority + ticks) : 0),
    highest ((active_mm == running_active_mm) ? 1:0)
  }
```

Deklarationsteil

Bsp. Policy-File: Linux 2.4 Schedulerverhalten(2)

```
handler (event e) {

  [...]

  On block.* {
    e.target => blocked;
    if (empty(READY)) {running_active_mm = NULL;}
  }

  On unblock.* {
    if (e.target in blocked) {
      if ((!empty(running)) && (e.target > running))
         {running => ready; }
      e.target => ready;
    }
  }

 [...]

  On system.clocktick {
      if (running.policy != BOSSA_SCHED_FIFO) {
        running.ticks--;
        if (running.ticks <= 0) { // intslice expired ?
          running.ticks = (((running.priority)>>2)+1); // reload ticks
          if (running.policy == BOSSA_SCHED_RR) {
            running => ready;
          }
          else {
            running => expired;       // preempt running
          }
        }
      }
}
```

Eventhandler (Ausschnitt)

Bsp. Policy-File: Linux 2.4 Schedulerverhalten(3)

```
Interface = {

    void attach (process p, policy_t policy, int rt_priority, int niceval) {
        // insert checks here...
      p.policy = policy;
      p.rt_priority = rt_priority;
      p.priority = 20 - niceval;
      p.ticks = (p.priority>>2)+1;
      p => ready;
    }



    void detach (process p) {
      if (running_active_mm == p.active_mm) {
        running_active_mm = NULL;
      }
    }



    void set_priority(process p, int niceval) {
        // insert checks here
      p.priority = 20 - niceval;
    }
```

Interfacefunkionen (Ausschnitt)

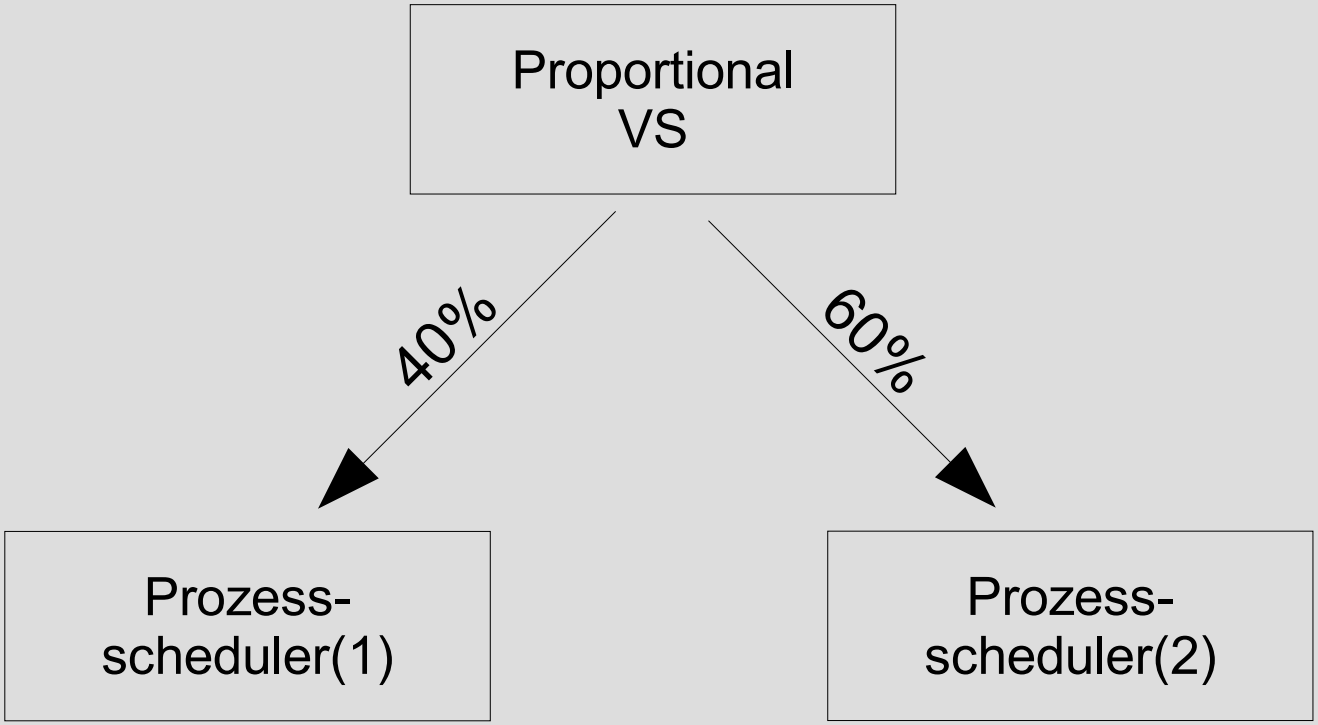
BOSSA
Zwischenresumé

- “Normaler” Scheduler mittels BOSSA relativ einfach zu realisieren
- Bisherige Konzepte im Prinzip für alle Zwecke ausreichend
- Aber: komplexere Schedulerverfahren leicht unüberschaubar

Virtuelle Scheduler

- Normalfall: Nur 1 Prozess-scheduler
- Bossa: Konzept des “virtual scheduler” (VS)
- Aufbau von komplexen (Baum-)Hierarchien: VS scheduled zwischen anderen Schedulern
- Implementierung unterschiedlichster Prioritätsklassen nur mittels VS
- Bsp. VS: Proportional-Scheduler

Virtuelle Scheduler(2)
Bsp. Proportional virtual scheduler



Virtuelle Scheduler(3)
Implementierung:

- Keyword `virtual_scheduler`
- Weiterleitung von Prozess-Events an Kindscheduler → `forwardImmediate();`
- Schnittstelle für attachen des Kindschedulers mit entspr. Parametern

Virtuelle Scheduler(4)
Codebeispiel:

Ausschnitt Eventhandler (proportional VS)

```
On unblock.*, block.*, yield.*, process.* {
  next(e.target) => forwardImmediate();
}


On system.clocktick {
  scheduler s = running;
  s => forwardImmediate ();
  s.ticks--;
  if (!(s in BLOCKED) && s.ticks <= 0) {so suspend the scheduler
    s.ticks = (s.proportion * period) / MaxProp;
    s => expired;
  }
}
```

BOSSA
Verwendung der Schedulerhierarchie

- Laden der Scheduler-Kernelmodule
- Schedulerhierarchie aufbauen
  → (manager, `/dev/BOSSA/,...`)
- Anwendungen in gewünschten Scheduler einhängen
  → (renice, `myScheduler_attach(),...`)

BOSSA
Kritische Betrachtung:

Pro:

- **Schedulerentwicklung für quasi “Jedermann”**
- **Forschung:**
  Bequemens Testen neuer Schedulerstrategien
- **Praxis:**
  Leichtes Adaptieren des Schedulers für eigene spezielle Ansprüche

Kontra:

- **Performanceeinbussen (max. 10%, meist aber weniger)**
- **Speziell adaptierter BOSSA-Kern erforderlich (Updaterisiko)**

Quellen und weitere Informationen

Dokumentation, Codebeispiele, related papers:

Devil Projekthomepage

`http://compose.labri.fr/prototypes/devil/`

Bossa Projekthomepage

`http://www.emn.fr/x-info/bossa/`