The Bossa Framework for Scheduler Development

Julia Lawall, DIKU, University of Copenhagen

Gilles Muller, OBASCO Group, Ecole des Mines de Nantes-INRIA, LINA

Target domain: Kernel process scheduling

Process scheduling: How an OS selects a process for the CPU.

- Many scheduling policies (round-robin, RM, EDF, etc.).
- No policy is perfect for all applications.

Implementing a scheduler requires:

- Understanding the scheduling policy.
- Understanding the target OS.
  - Any error can crash the machine.

Our proposal: Bossa

A Run-Time System for integrating scheduling policies into a legacy OS

- Generates event notifications.
- Defines a model of kernel scheduling requirements.

A Domain-Specific Language for implementing scheduling policies.

- DSL: A language dedicated to a particular domain.
- Provides high-level domain-specific abstractions that
  - Hide technical details.
  - Simplify programming.
  - Enable verifications, optimizations.

Overview

- The process scheduling problem.
- The Bossa DSL.
- Preparing Linux for use with Bossa.
- Verification.
- Performance.
- Conclusion, ongoing work.

The process scheduling problem

CPU:

Other processes:

The process scheduling problem

CPU:

Other processes:

A process arrives.

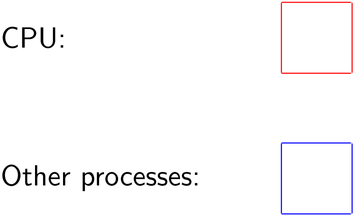
The process scheduling problem

CPU:

Other processes:

The process is elected.

The process scheduling problem



CPU:

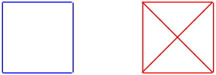
Other processes:

Another process arrives.

The process scheduling problem

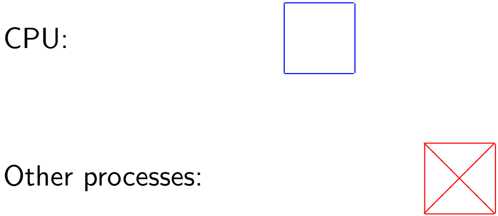
CPU:

Other processes:



The red process blocks.

The process scheduling problem

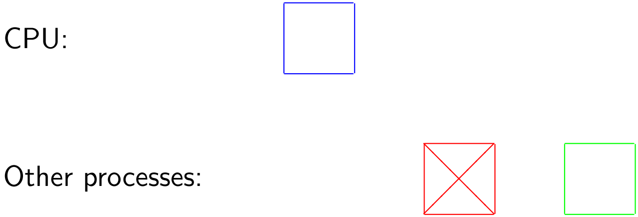


CPU:

Other processes:

The blue process is elected.

The process scheduling problem

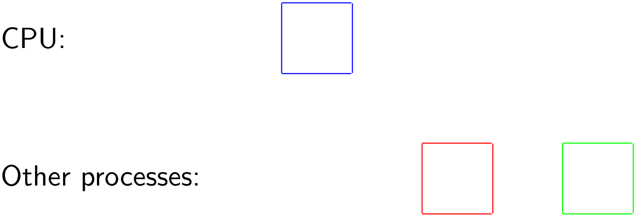


CPU:

Other processes:

Another process arrives.

The process scheduling problem



CPU:

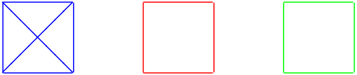
Other processes:

The red process unblocks.

The process scheduling problem

CPU:

Other processes:



The blue process blocks.

The process scheduling problem

CPU:

Other processes:

Which process is elected next?

Scheduling concepts

- Process states (running, ready, blocked, etc.).
- Election criteria.
- OS events (blocking, unblocking, etc.).

The Bossa DSL, by example

```
scheduler EDF = {








  states = {
             running
             ready
             blocked
             computation_ended
                terminated
  }


}
```

The Bossa DSL, by example

```
scheduler EDF = {

  states = {
    RUNNING running
    READY   ready
    BLOCKED blocked
    BLOCKED computation_ended
    TERMINATED terminated
  }

}
```

The Bossa DSL, by example

```
scheduler EDF = {









  states = {
    RUNNING running : process;
    READY   ready   : select queue;
    BLOCKED blocked : queue;
    BLOCKED computation_ended : queue;
    TERMINATED terminated;
  }


}
```

The Bossa DSL, by example

```
scheduler EDF = {







  states = {
    RUNNING running : process;
    READY   ready   : select queue;
    BLOCKED blocked : queue;
    BLOCKED computation_ended : queue;
    TERMINATED terminated;
  }
  ordering_criteria = {                         }

}
```

The Bossa DSL, by example

```
scheduler EDF = {
  process = {
    time  period;
    time  wcet;
    time  current_deadline;
    timer period_timer;
  }
  states = {
    RUNNING running : process;
    READY   ready   : select queue;
    BLOCKED blocked : queue;
    BLOCKED computation_ended : queue;
    TERMINATED terminated;
  }
  ordering_criteria = {                        }

}
```

The Bossa DSL, by example

```
scheduler EDF = {
  process = {
    time  period;
    time  wcet;
    time  current_deadline;
    timer period_timer;
  }
  states = {
    RUNNING running : process;
    READY   ready   : select queue;
    BLOCKED blocked : queue;
    BLOCKED computation_ended : queue;
    TERMINATED terminated;
  }
  ordering_criteria = { lowest current_deadline }

}
```

The Bossa DSL, by example

```
scheduler EDF = {
  process = {
    time  period;
    time  wcet;
    time  current_deadline;
    timer period_timer;
  }
  states = {
    RUNNING running : process;
    READY   ready   : select queue;
    BLOCKED blocked : queue;
    BLOCKED computation_ended : queue;
    TERMINATED terminated;
  }
  ordering_criteria = { lowest current_deadline }
  handler (event e) { ... }
}
```

Bossa event handlers

```
handler (event e) {
  On block.*

  On unblock.preemptive







  On bossa.schedule



  ...
}
```

Bossa event handlers

```
handler (event e) {
  On block.* { e.target => blocked; }

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

  On bossa.schedule {
    select() => running;
  }
  ...
}
```

Other features

Timers

- Used in EDF to wake a process at the beginning of its period.

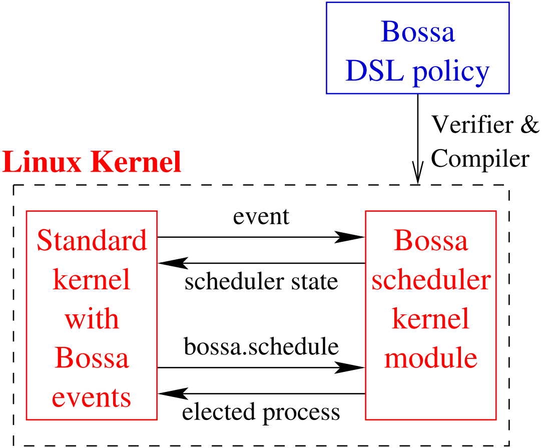
Interface functions

- Used in EDF to pause a process at the end of its computation.

Attach/detach functions

- Used in EDF to check schedulability.

Preparing Linux for use with Bossa



Problem: adding Bossa event notifications to Linux

Traditional approach: Modify code and create a patch file.

- Tedious and error-prone.
- Only applies to one version of the OS.
- Non-modular.

Problem: adding Bossa event notifications to Linux

Our approach:

Aspect-Oriented Programming (AOP)

- Where: e.g., around call to `try_to_wakeup`.
- What: e.g., call Bossa event `rts_unblock`.
- Independent of line numbers and code details.
- Portable across multiple versions.

Components

- Describe the interface between the OS and the Bossa policy.
- Interface augmented with aspect rules.

Limitations of traditional aspect systems

Traditional aspect systems:

- Put code before, after, and around functions and variables.
- No mechanism for referring to code sequences.

Linux `schedule` function:

```
schedule (void) {
  spin_lock_irq(&runqueue_lock);
  ... process election ...
  spin_unlock_irq(&runqueue_lock);
  ... context switch ...
}
```

Limitations of traditional aspect systems

Traditional aspect systems:

- Put code before, after, and around functions and variables.
- No mechanism for referring to code sequences.

Linux schedule function:

```
schedule (void) {
  spin_lock_irq(&runqueue_lock);
  ... process election ...
  spin_unlock_irq(&runqueue_lock);
  ... context switch ...
}
```

Extend AOP with Temporal Logic to describe code sequences.

Implemented using CIL and applied to Linux 2.4.18 and 2.4.24.

Verification

A Bossa policy must respect the scheduling requirements of the target OS.

Event types:

- Model of OS behavior.
- Created by the OS expert who integrates Bossa event notifications into the OS.
- Example: `unblock.preemptive` for Linux.

```
[tgt in BLOCKED] -> [tgt in READY]

[p in RUNNING, tgt in BLOCKED] -> [[p,tgt] in READY]

[tgt in RUNNING] -> []

[tgt in READY] -> []
```

Event type checking

`unblock.preemptive` for Linux.

```
[tgt in BLOCKED] -> [tgt in READY]
[p in RUNNING, tgt in BLOCKED] -> [[p,tgt] in READY]
[tgt in RUNNING] -> []
[tgt in READY] -> []
```

Example definition:

```
On unblock.preemptive {

    e.target => ready;
    if (!empty(running) && e.target > running) {
      running => ready;
    }

}
```

Incorrect for Linux!

Event type checking

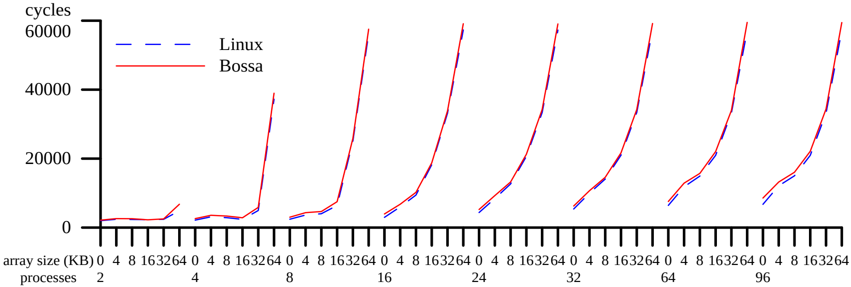
unblock.preemptive for Linux.

```
[tgt in BLOCKED] -> [tgt in READY]

[p in RUNNING, tgt in BLOCKED] -> [[p,tgt] in READY]

[tgt in RUNNING] -> []

[tgt in READY] -> []
```

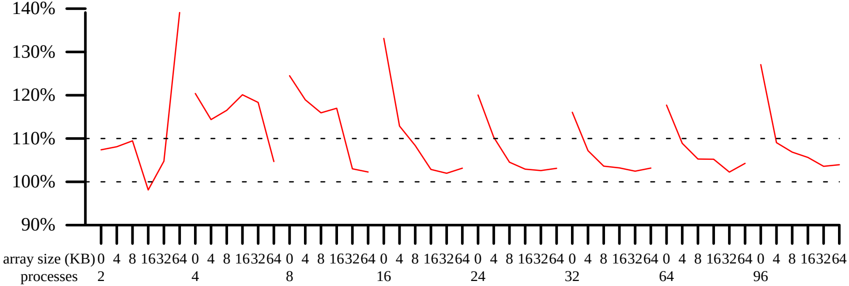
Example definition:

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

Performance: lat_ctx, context switch time



Performance: lat_ctx, increase as compared to Linux



Conclusions

Specialized process schedulers needed for demanding applications, but schedulers are not easy to implement in existing OSes.

Bossa provides:

- A DSL to ease the programming of scheduling policies.
- A Run-Time System implementing a scheduling interface.
- Verifications checking that a scheduling policy meets OS requirements.

Availability:

- Several versions of Linux 2.4.
- Teaching lab, based on Knoppix.

Ongoing work

- Modular Bossa [GPCE05], with a modular type system.
- Development of verified schedulers within the B framework.
- BossaBox: PVR with programmable scheduling.
- Coccinelle: automated support for device driver evolution [ACP4IS].

More information

- Framework [HASE 2005]
- Aspects and components [SIGOPS 2004]
- Verification [GPCE 2004]
- Generalization to a scheduler hierarchy [PEPM 2004]

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