Rethinking CPU scheduling for dynamic workloads on Sculpt OS

Johannes Schlatow and Stefan Kalkowski
<$firstname.$lastname@genode-labs.com>

Introduction

What is Sculpt OS?

- showcase for Genode OS Framework as Desktop OS
  - component-based OS framework
  - hierarchical distribution of resources
  - supports several microkernels
- official Sculpt OS image (`x86_64`) uses NOVA microhypervisor
- custom kernel (base-hw) primarily for ARM and test-driving new ideas

Sculpt OS: User perspective

Sculpt user assigns for each component:

- four scheduling groups with different latency expectations
  - **driver** – *latency-sensitive device drivers*
  - **multimedia** – *audio, video, latency-sensitive parts of UI*
  - **default** – *UI, desktop apps, computing*
  - **background** – *best effort*
- mapped to scheduling priorities of underlying microkernel
- rogue driver leads to starvation of lower-priority

→ **fixed-priority scheduling may lead to starvation**

Quota-based scheduling on custom kernel

Scheduling in Genode's custom kernel (base-hw) since 2014

- quota- and priority-based scheduling with radical degradation to round-robin
- mitigates rogue threads (e. g. broken device drivers)
- enables hierarchical distribution of CPU quota
- requires tuning for workload and target platform

Quota-based scheduling on custom kernel

Scheduling in Genode's custom kernel (base-hw) since 2014

- quota- and priority-based scheduling with radical degradation to round-robin
- mitigates rogue threads (e. g. broken device drivers)
- enables hierarchical distribution of CPU quota
- requires tuning for workload and target platform

→ **no adequate solution for dynamic workloads**

Scheduling requirements for dynamic workloads

Take a step back: What do we need?

- fairness and adjustable latency
  - fair (proportional) share of CPU → prevents starvation
  - some threads prefer low latency
  - other threads do not care much about latency (best-effort and high-throughput)
- ease of configuration
- robustness against misconfiguration and workload changes

Scheduling requirements for dynamic workloads

Take a step back: What do we need?

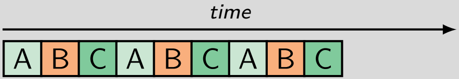
- fairness and adjustable latency
  - fair (proportional) share of CPU → prevents starvation
  - some threads prefer low latency
  - other threads do not care much about latency (best-effort and high-throughput)
- ease of configuration
- robustness against misconfiguration and workload changes

→ **revise scheduling for custom kernel**

Fair scheduling

Round-Robin

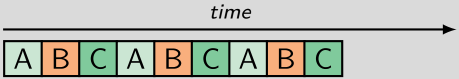
- each thread gets an equal share
- fixed time-slice length



Fair scheduling

Round-Robin

- each thread gets an equal share
- fixed time-slice length



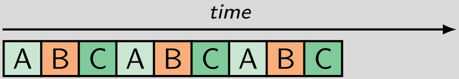
Weighted Round-Robin

- each thread is assigned a weight
- CPU share proportional to weight

Fair scheduling

Round-Robin

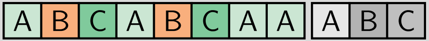
- each thread gets an equal share
- fixed time-slice length



A	B	C	A	B	C	A	B	C

Weighted Round-Robin

- each thread is assigned a weight
- CPU share proportional to weight
- different interleaving schemes
  - weights: A=4, B=2, C=2



A	B	C	A	B	C	A	A

A	B	C

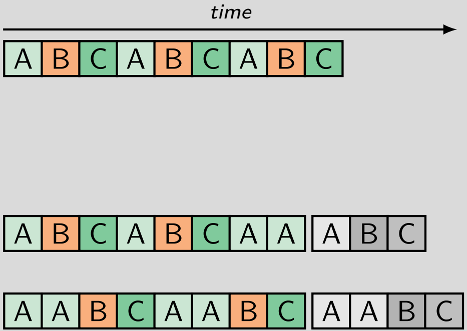
Fair scheduling

Round-Robin

- each thread gets an equal share
- fixed time-slice length

Weighted Round-Robin

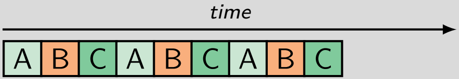
- each thread is assigned a weight
- CPU share proportional to weight
- different interleaving schemes
    - weights: A=4, B=2, C=2



Fair scheduling

Round-Robin

- each thread gets an equal share
- fixed time-slice length

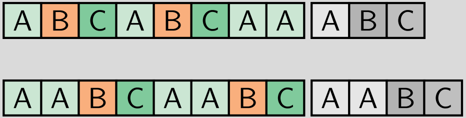


time

A B C A B C A B C

Weighted Round-Robin

- each thread is assigned a weight
- CPU share proportional to weight
- different interleaving schemes
  - weights: A=4, B=2, C=2
- round-based implementation
- not trivial for dynamic workloads



A B C A B C A A | A B C

A A B C A A B C | A A B C

Fair scheduling for dynamic workloads

Virtual-time scheduling

- each thread has a weight and a virtual time
- scheduler picks thread with minimum virtual time

$$\text{virtual time} += \frac{\Delta\text{real time}}{\text{weight}}$$

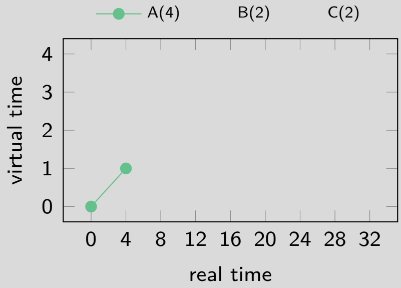
Fair scheduling for dynamic workloads

Virtual-time scheduling

- each thread has a weight and a virtual time
- scheduler picks thread with minimum virtual time

$$\text{virtual time} \mathrel{+}= \frac{\Delta\text{real time}}{\text{weight}}$$

Example: run each thread for 4 units of real time



A

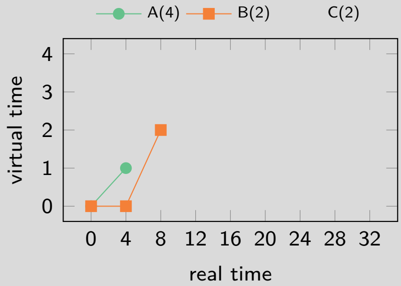
Fair scheduling for dynamic workloads

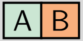
Virtual-time scheduling

- each thread has a weight and a virtual time
- scheduler picks thread with minimum virtual time

$$\text{virtual time} \mathrel{+}= \frac{\Delta\text{real time}}{\text{weight}}$$

Example: run each thread for 4 units of real time





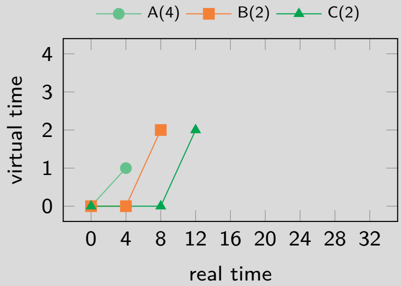
Fair scheduling for dynamic workloads

Virtual-time scheduling

- each thread has a weight and a virtual time
- scheduler picks thread with minimum virtual time

$$\text{virtual time} \mathrel{+}= \frac{\Delta\text{real time}}{\text{weight}}$$

Example: run each thread for 4 units of real time



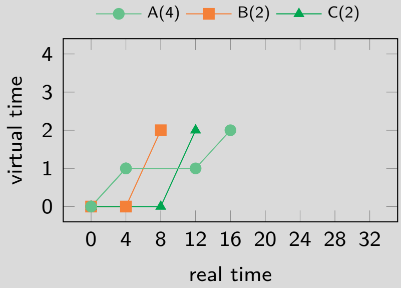
Fair scheduling for dynamic workloads

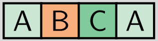
Virtual-time scheduling

- each thread has a weight and a virtual time
- scheduler picks thread with minimum virtual time

$$\text{virtual time} \mathrel{+}= \frac{\Delta\text{real time}}{\text{weight}}$$

Example: run each thread for 4 units of real time





A B C A

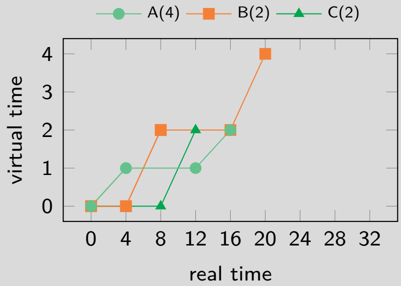
Fair scheduling for dynamic workloads

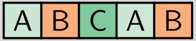
Virtual-time scheduling

- each thread has a weight and a virtual time
- scheduler picks thread with minimum virtual time

$$\text{virtual time} \mathrel{+}= \frac{\Delta\text{real time}}{\text{weight}}$$

Example: run each thread for 4 units of real time





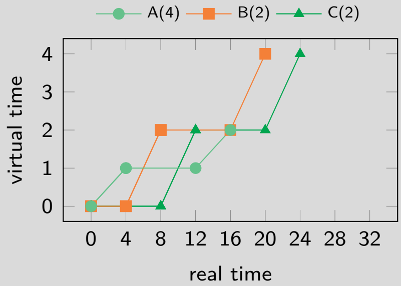
Fair scheduling for dynamic workloads

Virtual-time scheduling

- each thread has a weight and a virtual time
- scheduler picks thread with minimum virtual time

$$\text{virtual time} \mathrel{+}= \frac{\Delta\text{real time}}{\text{weight}}$$

Example: run each thread for 4 units of real time



A	B	C	A	B	C

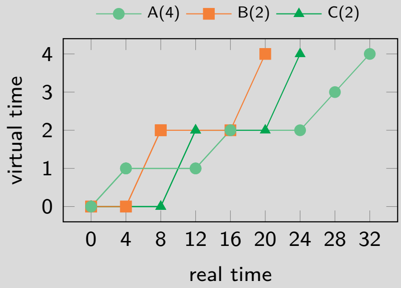
Fair scheduling for dynamic workloads

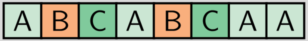
Virtual-time scheduling

- each thread has a weight and a virtual time
- scheduler picks thread with minimum virtual time

$$\text{virtual time} \mathrel{+}= \frac{\Delta\text{real time}}{\text{weight}}$$

Example: run each thread for 4 units of real time




A B C A B C A A

Challenges with virtual-time scheduling

Open questions

- How to assign **weights** to threads?
- How to determine **time-slice** length?
- How to deal with **sleeping** threads?
- How to tune threads for low **latency**?

Challenges with virtual-time scheduling

Open questions

- How to assign **weights** to threads?
- How to determine **time-slice** length?
- How to deal with **sleeping** threads?
- How to tune threads for low **latency**?

Existing solutions

- EEVDF (Earliest Eligible Virtual Deadline First), new replacement for Linux' CFS
- BVT (Borrowed Virtual Time) ← **our basis for new scheduler**

Challenges with virtual-time scheduling

Open questions

- How to assign **weights** to threads?
- How to determine **time-slice** length?
- How to deal with **sleeping** threads?
- How to tune threads for low **latency**?

Existing solutions

- EEVDF (Earliest Eligible Virtual Deadline First), new replacement for Linux' CFS
- BVT (Borrowed Virtual Time) ← **our basis for new scheduler**

Approach

1. simulate basic scenarios to experiment with parameters and options
2. implement in custom kernel and evaluate

Adapting virtual-time scheduling for Sculpt OS

How to assign weights to threads?

$$\text{proportional share} = \frac{\text{sum of weights}}{\text{thread weight}}$$

- adding threads changes proportional share
  - e. g.: CPU share for **driver** threads shrinks with number of threads in **default**
- re-assigning weights of running threads complicates implementation

Adapting virtual-time scheduling for Sculpt OS

How to assign weights to threads?

$$\text{proportional share} = \frac{\text{sum of weights}}{\text{thread weight}}$$

- adding threads changes proportional share
  - e. g.: CPU share for **driver** threads shrinks with number of threads in **default**
- re-assigning weights of running threads complicates implementation

Solution: hierarchical scheduling

- fixed proportional share for groups (hard-coded guesstimate)
- equal weights within each group
- relieves user from weight assignment

Dynamic time slices

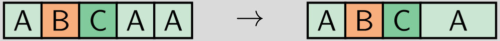
How to determine time-slice length?

- **options**: constant / thread property / derived from weight

Dynamic time slices

How to determine time-slice length?

- **options**: constant / thread property / derived from weight
- **idea from BVT**: eliminate context switches to the same thread

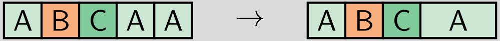


A B C A A → A B C A

Dynamic time slices

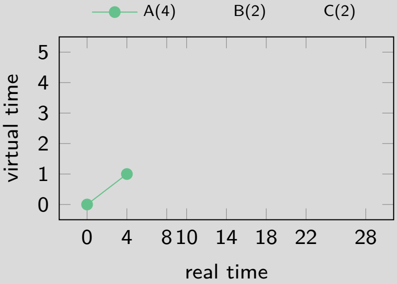
How to determine time-slice length?

- **options**: constant / thread property / derived from weight
- **idea from BVT**: eliminate context switches to the same thread



A B C A A → A B C A

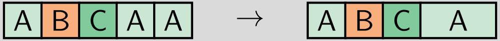
- catch up with lowest virtual time **plus** execute for a constant (real) time longer



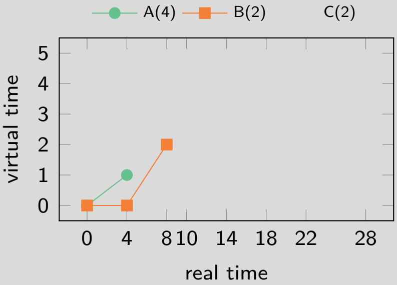
Dynamic time slices

How to determine time-slice length?

- **options**: constant / thread property / derived from weight
- **idea from BVT**: eliminate context switches to the same thread



- catch up with lowest virtual time **plus** execute for a constant (real) time longer



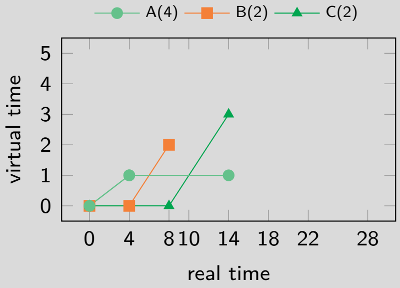
Dynamic time slices

How to determine time-slice length?

- **options**: constant / thread property / derived from weight
- **idea from BVT**: eliminate context switches to the same thread

A	B	C	A	A	→	A	B	C	A

- catch up with lowest virtual time **plus** execute for a constant (real) time longer



A	B	C

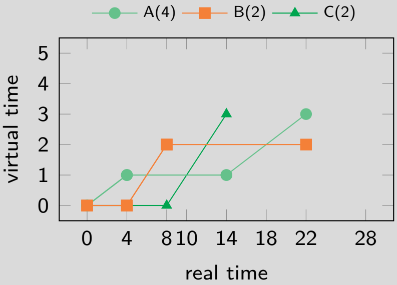
Dynamic time slices

How to determine time-slice length?

- **options**: constant / thread property / derived from weight
- **idea from BVT**: eliminate context switches to the same thread

A | B | C | A | A → A | B | C | A

- catch up with lowest virtual time **plus** execute for a constant (real) time longer

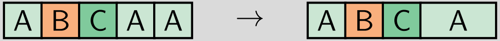


A | B | C | A

Dynamic time slices

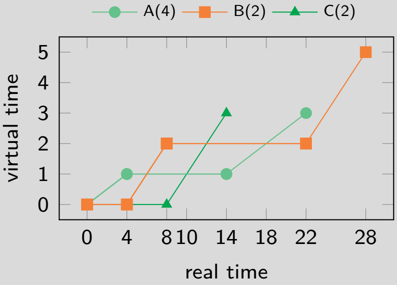
How to determine time-slice length?

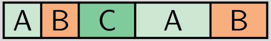
- **options**: constant / thread property / derived from weight
- **idea from BVT**: eliminate context switches to the same thread



A B C A A → A B C A

- catch up with lowest virtual time **plus** execute for a constant (real) time longer

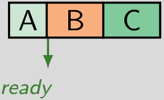




A B C A B

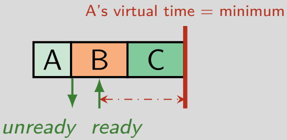
Sleeping policy

Let's take a nap!



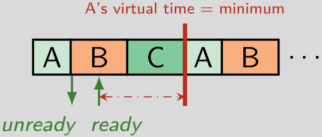
Sleeping policy

Let’s take a nap!



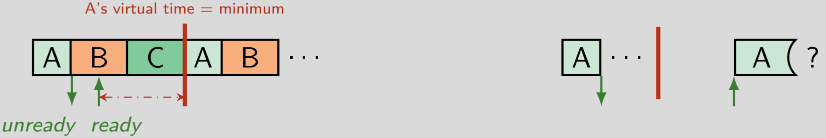
Sleeping policy

Let's take a nap!



Sleeping policy

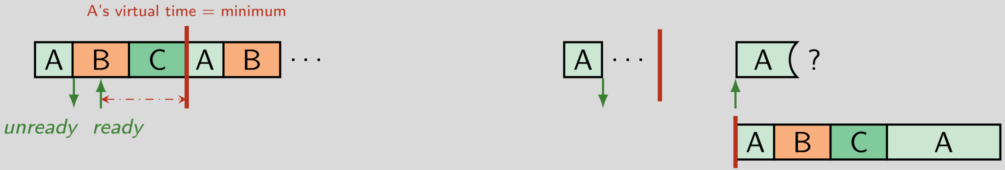
Let's take a nap!



- sleeping time must not affect time-slice length

Sleeping policy

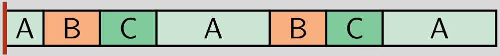
Let's take a nap!



- sleeping time must not affect time-slice length
- **BVT**: increase virtual time to minimum virtual time when thread becomes ready
- proportional share is only maintained for active threads

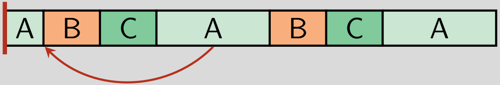
Latency tuning

How to tune threads for low latency?



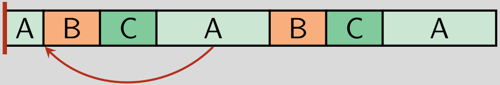
Latency tuning

How to tune threads for low latency?



Latency tuning

How to tune threads for low latency?

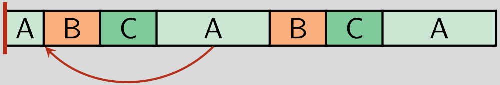


From BVT:

- each thread is assigned a *warp value*
- virtual time of a warping thread is reduced by its warp value
  - thread is preferred in scheduling decision
  - thread's time slice increases by *warp value * weight*

Latency tuning

How to tune threads for low latency?



From BVT:

- each thread is assigned a *warp value*
- virtual time of a warping thread is reduced by its warp value
  - thread is preferred in scheduling decision
  - thread's time slice increases by *warp value * weight*

In Sculpt:

- per-group warp value (hard-coded guesstimate)
- warp state of group derived from thread with lowest virtual time
- fixed time limit for each thread to run with warp enabled without sleeping

Configuration

Configuration

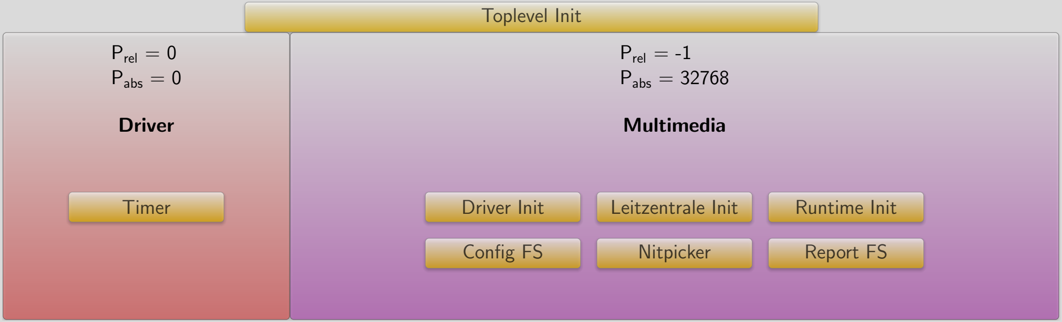
Hard-coded scheduling configuration

- four hard-coded scheduling groups

Group	Weight	Warp
driver	10	4.5ms
multimedia	5	4ms
default	5	2ms
background	1	0ms

- Sculpt's two-level priority hierarchy mapped to groups

Mapping Sculpt's component priorities to scheduling groups



Toplevel Init

P_{rel} = 0
P_{abs} = 0

Driver

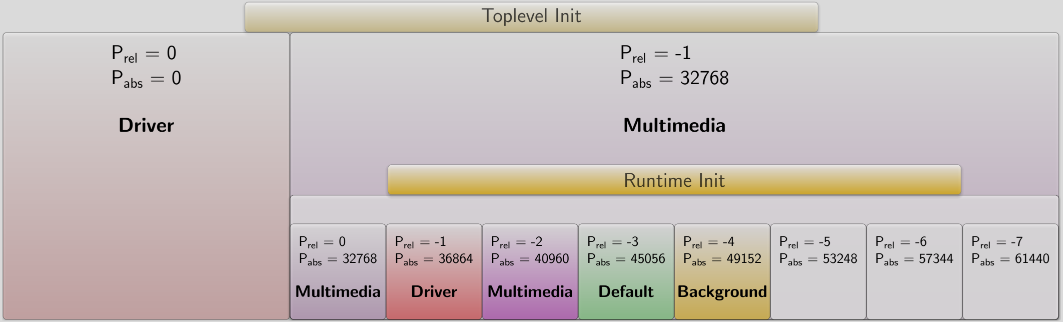
Timer

P_{rel} = -1
P_{abs} = 32768

Multimedia

Driver Init
Leitzentrale Init
Runtime Init
Config FS
Nitpicker
Report FS

Mapping Sculpt's component priorities to scheduling groups



Toplevel Init

$P_{rel} = 0$
$P_{abs} = 0$

Driver

$P_{rel} = -1$
$P_{abs} = 32768$

Multimedia

Runtime Init

$P_{rel} = 0$
$P_{abs} = 32768$
Multimedia

$P_{rel} = -1$
$P_{abs} = 36864$
Driver

$P_{rel} = -2$
$P_{abs} = 40960$
Multimedia

$P_{rel} = -3$
$P_{abs} = 45056$
Default

$P_{rel} = -4$
$P_{abs} = 49152$
Background

$P_{rel} = -5$
$P_{abs} = 53248$

$P_{rel} = -6$
$P_{abs} = 57344$

$P_{rel} = -7$
$P_{abs} = 61440$

Demo

Live demo

Summary

- implementation matches expectations
  - in benchmarks
  - in user experience on Sculpt
- side-effect: reworked base-hw internals
- minimal configuration burden for Sculpt OS users
- groups and group-parameters hard-coded but configurable in framework

Sculpt OS 25.10 image with base-hw for x86_64

`https://depot.genode.org/skalk/image/sculpt-pc-2026-01-29.img.xz`

References

GitHub Issue #5117 (Concept & Implementation)
https://github.com/genodelabs/genode/issues/5117

GitHub Issue #5604 (Evaluation)
https://github.com/genodelabs/genode/issues/5604

Borrowed-virtual-time (BVT) scheduling - Duda and Cheriton
https://dl.acm.org/doi/10.1145/319151.319169

Earliest Eligible Virtual Deadline First (EEVDF) - Stoica and Abdel-Wahab
https://people.eecs.berkeley.edu/~istoica/papers/eevdf-tr-95.pdf

FOSDEM 2017 talk on base-hw
https://archive.fosdem.org/2017/schedule/event/microkernel_kernel_library/