# Scheduling Priorities and FreeBSD: A Deep Dive (and Sweep)

Olivier Certner

Kumacom SARL

EuroBSDCon 2024

# Olivier Certner

- 🇫🇷
- CS professional for ∼20 years
- PhD in many-core parallel programming models
- Languages expert
  - Notably, C, Common Lisp, Ada and C++
- Developer, software architect, systems design
- Worked in the CAD and finance sectors
- Former CTO of small startups

# Involvement With FreeBSD

## Private

- Using FreeBSD since 2004
- Using it everywhere I can
- Maintaining small private changes (ports, userland, kernel).

## Public

- Since ~20 years: Sporadic bug reports and mails on lists
- Since ~4 years: Gradual increase in involvement
  - Maintaining a few ports
  - Reporting bugs in base and submitting patches
- Since a year: Working full time
  - Contractor for the FreeBSD Foundation since 2023/09
  - Committer since ~9 months (`olce@`)
  - Presented at AsiaBSDCon 2024

# Past and Current Other Work

- Login classes
- Process visibility
- Zenbleed mitigation
- Vnode recycling and ZFS ARC reclaim
  - PR 275594: The critical issue
  - Followups (WIP)
- `mac_do(4)`
  - Conceptual changes, make it robust
- `unionfs(4)`
  - Long term proposal
  - Review Jason Harmening's (`jah@`) last batch of fixes
- Reviews of others' work

# Project Goals

## Rationalize and Make Scheduling Priorities Robust

- Fix scheduling APIs bugs
  - Behavior
  - Security
- Decouple the implementation and interfaces
- Better POSIX compliance
  - In effect or in spirit
  - Except when poor or non-sensical
- Extend usefulness
  - Confine processes
  - Make timesharing's priority levels useful
  - Improve priority reporting

# AsiaBSDCon 2024's Paper Content

- Provide an exhaustive API reference for `rtprio(2)` and POSIX(.1b)
  - History of POSIX standard documents
- Mention differing platform's behaviors
- Expose old and new design choices
- Report on progress (back then)

# This Talk

1. Scheduling Policies
   - Background
   - `rtprio(2)`
   - POSIX(.1b)

# This Talk

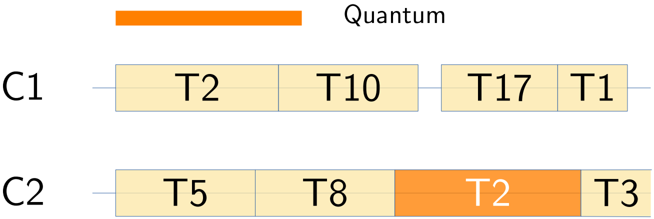
1. Scheduling Policies
   - Background
   - `rtprio(2)`
   - POSIX(.1b)
2. Impacting Changes
   - Timesharing Priority Levels
   - Scheduling Privileges
   - Real-Time Priorities
   - ULE

# Background

# Scheduling

## Decide

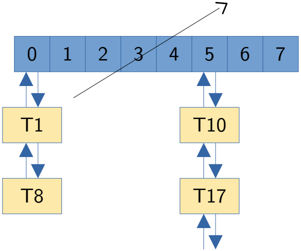
- Which runnable thread
- Runs on which CPU/core
- At which moment
- For which duration



# Priority-Based Models

## Classical Alternatives

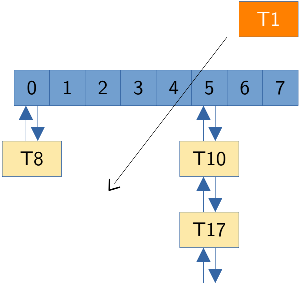
- Multi-level queue
  - Fixed-level assignment
- Multi-level feedback queue
  - Dynamic level changes
  - Based on behavior



# Priority-Based Models

## Classical Alternatives

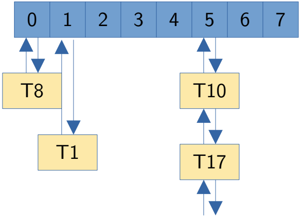
- Multi-level queue
  - Fixed-level assignment
- Multi-level feedback queue
  - Dynamic level changes
  - Based on behavior



# Priority-Based Models

## Classical Alternatives

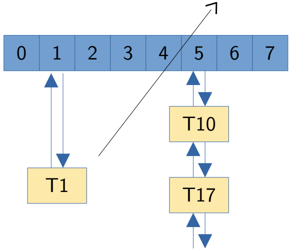
- Multi-level queue
  - Fixed-level assignment
- Multi-level feedback queue
  - Dynamic level changes
  - Based on behavior



# Priority-Based Models

## Classical Alternatives

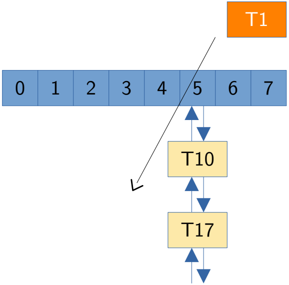
- Multi-level queue
  - Fixed-level assignment
- Multi-level feedback queue
  - Dynamic level changes
  - Based on behavior



# Priority-Based Models

## Classical Alternatives

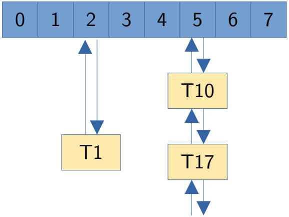
- Multi-level queue
  - Fixed-level assignment
- Multi-level feedback queue
  - Dynamic level changes
  - Based on behavior



# Priority-Based Models

## Classical Alternatives

- Multi-level queue
  - Fixed-level assignment
- Multi-level feedback queue
  - Dynamic level changes
  - Based on behavior



# FreeBSD Internal Model

## Selection Policies

- Realtime
  - Multi-level queue
  - Includes:
    - Interrupt (kernel) threads
    - Realtime user threads
    - Regular kernel threads
    - ULE: Interactive user threads
- Timeshare
  - Multi-level feedback queue
- Idletime
  - Multi-level queue

# rtprio(2)

# API Overview

## System calls

| | |
|---|---|
| `rtprio(2)` | Operate on some process (PID) or the current thread (0) |
| `rtprio_thread(2)` | Operate on some thread (TID) |

## Modes

| | |
|---|---|
| `RTP_LOOKUP` | Retrieve settings |
| `RTP_SET` | Set settings |

## Settings

| | |
|---|---|
| `type` | Scheduling type/class |
| `prio` | Priority level within the class<br>• Higher number means lower priority |

# Scheduling Types/Classes 1

From highest to lowest priority:

- Interrupt threads

  type `RTP_PRIO_ITHD`
  prio ?

- Realtime user threads

  type `RTP_PRIO_FIFO` or `RTP_PRIO_REALTIME`
  prio 0–31 with a caveat...

- Regular kernel threads

  type `RTP_PRIO_NORMAL`
  prio 0

# Scheduling Types/Classes 1

From highest to lowest priority:

- Interrupt threads

  type `RTP_PRIO_ITHD`
  prio Implementation dependent

- Realtime user threads

  type `RTP_PRIO_FIFO` or `RTP_PRIO_REALTIME`
  prio 0–31

- Regular kernel threads

  type `RTP_PRIO_KERNEL`
  prio Implementation dependent

# Scheduling Types/Classes 2

From highest to lowest priority:

- Timesharing threads

  type `RTP_PRIO_NORMAL` or `RTP_PRIO_TIMESHARE`

  prio Implementation dependent... and dynamic!

- Idle threads

  type `RTP_PRIO_IDLE`

  prio 0–31 with a caveat...

# Scheduling Types/Classes 2

From highest to lowest priority:

- Timesharing threads

  type RTP_PRIO_NORMAL or RTP_PRIO_TIMESHARE

  prio 0–40

- Idle threads

  type RTP_PRIO_IDLE

  prio 0–31

# POSIX(.1b)

# The Standard

## Differences to `rtprio(2)`

- Priority levels “reversed”
- Absolute priority scale
- Process vs. thread scheduling settings
  - Effect depends on scheduling contention scope

## As in `rtprio(2)`

- Non-negative priority numbers

# Reality Check 1

- Priority levels “reversed”

  But not on:

  - HP-UX

# Reality Check 1

- Priority levels “reversed”

  But not on:

  - HP-UX

- Non-negative priority numbers

  But not on:

  - illumos
  - NetBSD

# Reality Check 2

- Absolute priority scale

  Well, supposedly... but not on:

  - FreeBSD
  - OpenBSD
  - NetBSD
  - illumos
  - Linux

# Reality Check 2

- Absolute priority scale

  Well, supposedly... but not on:

  - FreeBSD
  - OpenBSD
  - NetBSD
  - illumos
  - Linux
- Process vs. thread scheduling settings
  - Support for system contention scope only
  - Process settings should have no effect
  - But, in surveyed variants, they are mapped to:
    - Either the "main" thread, or the calling thread
    - Or all process' threads

# Outline

1 Scheduling Policies
- Background
- `rtprio(2)`
- POSIX(.1b)

2 Impacting Changes
- Timesharing Priority Levels
- Scheduling Privileges
- Real-Time Priorities
- ULE

# Timesharing Priority Levels

# Dynamic Levels

```
$ ./set_rtprio $$ NORMAL 10
Current priority:  0.
RT prio:  Type:  RTP_PRIO_NORMAL, prio:  10.
```

# Dynamic Levels

```
$ ./set_rtprio $$ NORMAL 10
Current priority:  0.
RT prio:  Type:  RTP_PRIO_NORMAL, prio:  10.
$ ./prio $$
Current priority:  0.
RT prio:  Type:  RTP_PRIO_NORMAL, prio:  0.
```

# Dynamic Levels

```
$ ./set_rtprio $$ NORMAL 10
Current priority:  0.
RT prio:  Type:  RTP_PRIO_NORMAL, prio:  10.
$ ./prio $$
Current priority:  0.
RT prio:  Type:  RTP_PRIO_NORMAL, prio:  0.
$ ./prio $$
Current priority:  0.
RT prio:  Type:  RTP_PRIO_NORMAL, prio:  1.
```

# Other Problems

- Tied to internal priority levels
  - Actual range: 0 – `PRI_MAX_TIMESHARE-PRI_MIN_TIMESHARE`
  - Changed in FreeBSD 14, may change again.
- `rtprio(2)` and POSIX(.1b) inconsistencies
  - `rtprio(2)` tries to set the internal priority
  - POSIX(.1b) just completely ignores the passed level

# Switch to Fixed Values

- But which ones?

# Switch to Fixed Values

- Readily available: Nice values

# Switch to Fixed Values

- Readily available: Nice values
    - But then, per-thread

# Switch to Fixed Values

- Readily available: Nice values
  - But then, per-thread
- POSIX compliance?
  - Absolute priority levels
  - Mapping to a process' nice value

# Switch to Fixed Values

- Readily available: Nice values
  - But then, per-thread
- POSIX compliance?
  - Absolute priority levels
  - Mapping to a process' nice value
- Backwards compatibility?
  - Settings will suddenly have an effect
  - Error returned on inconsistent parameters
  - Applications survey

# Scheduling Privileges

# Privilege Check Reminder

- Fine-grained privileges
  - Constants starting with `PRIV_`
  - Some examples: `PRIV_VFS_READ`, `PRIV_VFS_WRITE`
- Root has all privileges
  - Except in jails
- See full API description at `priv(9)`
  - `priv_check_cred()`
  - `priv_check()`

# Scheduling Privileges

## Initial List

- `PRIV_SCHED_SETPRIORITY`
- `PRIV_SCHED_RTPRIO`
- `PRIV_SCHED_IDPRIO`
- `PRIV_SCHED_SETPOLICY`
- `PRIV_SCHED_SET`
- `PRIV_SCHED_SETPARAM`

# Scheduling Privileges

## New List

- PRIV_SCHED_SETPRIORITY, PRIV_SCHED_RAISEPRIO
- PRIV_SCHED_RTPRIO
- PRIV_SCHED_IDPRIO
- PRIV_SCHED_SETPOLICY
- PRIV_SCHED_SET
- PRIV_SCHED_SETPARAM

# mac_priority(4)

- Users in group realtime can use realtime classes
  - Grants PRIV_SCHED_RTPRIO and PRIV_SCHED_SETPOLICY
  - Will also grant PRIV_SCHED_RAISEPRIO
- Users in group idletime can use the idletime class
  - Grants PRIV_SCHED_IDPRIO

# Real-Time Priorities

# POSIX XSH 2.8.4

*Conforming implementations shall provide a priority range of at least 32 priorities for this policy.*

(In both the `SCHED_FIFO` and `SCHED_RR` sections.)

# Are We Complying?

- `RTP_PRIO_MIN` is 0
- `RTP_PRIO_MAX` is 31
- (Added new static assertion.)
- What could possibly go wrong?

# Priority Levels Conflation

## FreeBSD Runqueue

- Has only 64 distinct levels (“queues”) vs. 256 priority levels
- Priority $P$ mapped to queue number $P/4$
    - 4 priorities per queue
- All threads on a single queue treated the same
- 1:1 `RTP_PRIO_REALTIME` levels ↔ internal priorities

# Priority Levels Conflation

## FreeBSD Runqueue

- Has only 64 distinct levels (“queues”) vs. 256 priority levels
- Priority $P$ mapped to queue number $P/4$
  - 4 priorities per queue
- All threads on a single queue treated the same
- 1:1 `RTP_PRIO_REALTIME` levels ↔ internal priorities

⇒ 4 consecutive `RTP_PRIO_REALTIME`’s levels treated the same!

# Switch to 1:1 Internal Priorities ↔ Queue's Levels

- Use a real 256-queue runqueue

# Switch to 1:1 Internal Priorities ↔ Queue's Levels

- Use a real 256-queue runqueue
  - Ensures distinct real 32 levels for realtime

# Switch to 1:1 Internal Priorities ↔ Queue's Levels

- Use a real 256-queue runqueue
  - Ensures distinct real 32 levels for realtime
  - Disturbs ULE's behavior...

# ULE

# A Tale of Three Runqueues

- One runqueue per selection policy
- Timesharing threads
  - Interactive ones ⇒ Real-time selection policy
  - Batch ones ⇒ Timesharing selection policy
- Timesharing selection policy (TSP)
  - Internal priority range has 88 values (136–223)
  - Mapped to 64 levels
  - Computed based on:
    - %CPU contribution ($[0; 47]$)
    - Nice contribution ($[0; 40]$)
  - Moving enqueue/dequeue offsets!

# The “Calendar” Queue

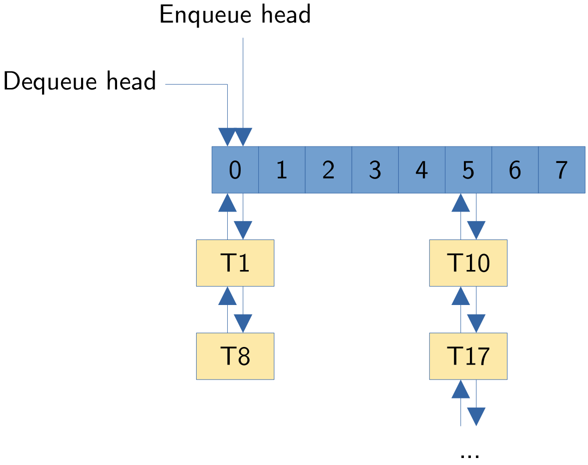
## Goals

- Anti-starvation and fairness
- Stay $O(1)$

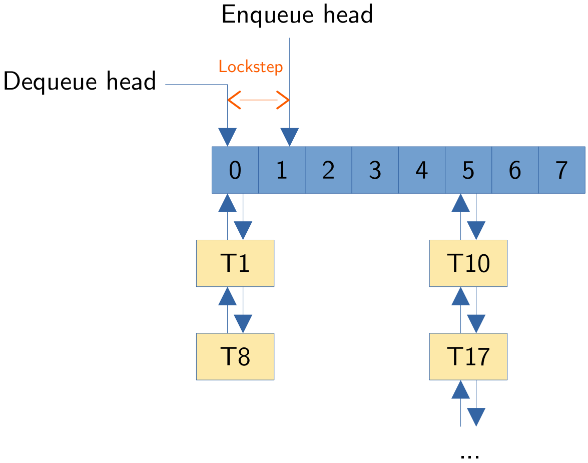
## Principles

- Mimicks 4BSD’s decay w/o updating priorities
- TSP’s runqueue treated as a circular queue
- Enqueue and dequeue heads move in locksteps
- Enqueue head advances by 1 at each tick (when it can)

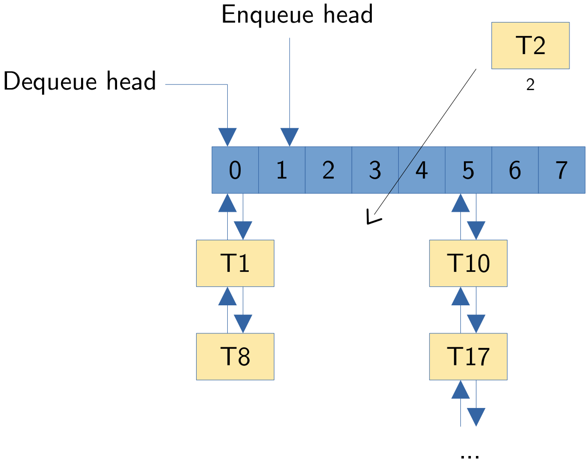
# The Calendar Queue: Example



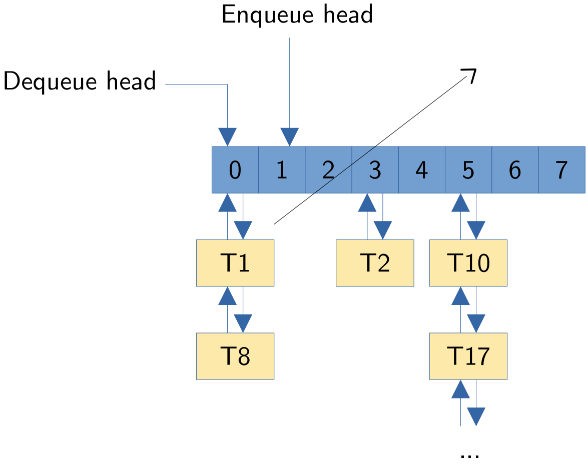
# The Calendar Queue: Example



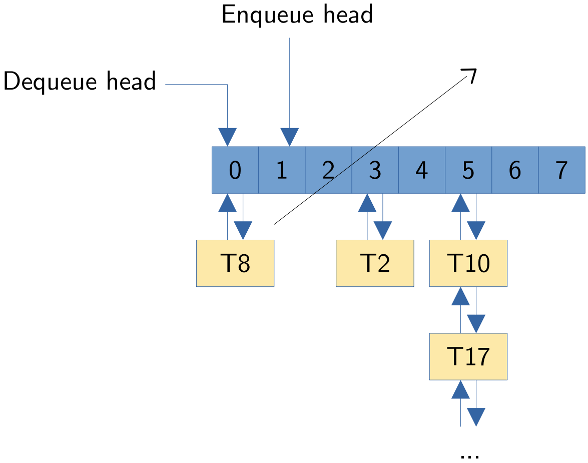
# The Calendar Queue: Example



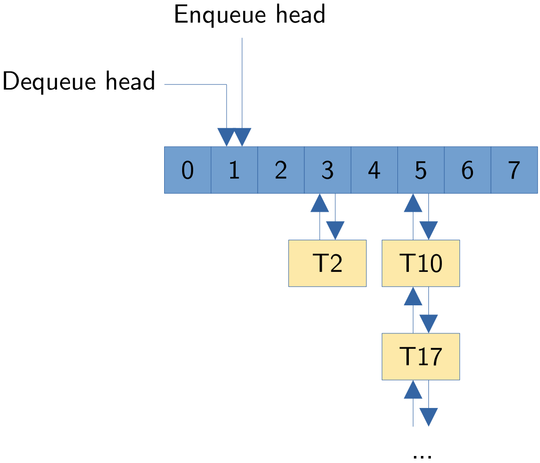
# The Calendar Queue: Example



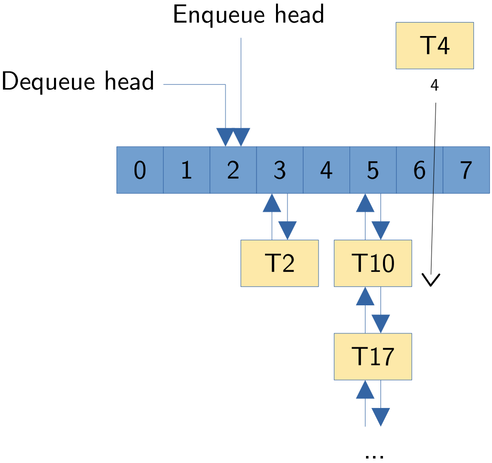
# The Calendar Queue: Example



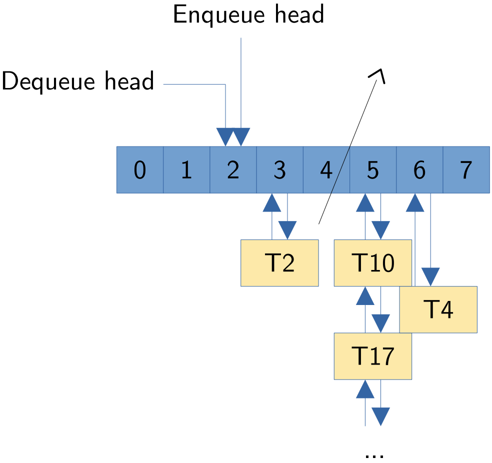
# The Calendar Queue: Example



# The Calendar Queue: Example



# The Calendar Queue: Example



# Now With One Runqueue

## What's Changed

- A range reserved for each selection policy
- Timesharing selection policy (TSP)
    - Range now has 109 values (115–223)
        - Use up freed values (no more folding)
        - Preserve the ratio of nice to CPU contribution
    - Mapped 1:1 to 109 queues
    - Preserve heads progression
        - Must return to initial position after 64 ticks
        - So, move it by 2 each tick, except for 1 tick out of 4 ($7/4 \approx 109/64$)

# Tests

- Run two threads with different nice values
- Compare the %CPU of the meanest thread
- They change at most by 1.15% and on average by 0.46%
- Relative: At most 2%, average 0.78%

# Tests

- Run two threads with different nice values
- Compare the %CPU of the meanest thread
- They change at most by 1.15% and on average by 0.46%
- Relative: At most 2%, average 0.78%

⇒ We seem to be good

# Tests

- Run two threads with different nice values
- Compare the %CPU of the meanest thread
- They change at most by 1.15% and on average by 0.46%
- Relative: At most 2%, average 0.78%

⇒ We seem to be good

- %CPU of nice -20 vs. nice 20: 66.7%...

# Tests

- Run two threads with different nice values
- Compare the %CPU of the meanest thread
- They change at most by 1.15% and on average by 0.46%
- Relative: At most 2%, average 0.78%

⇒ We seem to be good

- %CPU of nice -20 vs. nice 20: 66.7%...

⇒ Houston, we have a(n independent) problem!

To be continued!

# Some Other Achievements

- Factorize and fix priority translation
- Kernel drives almost everything
- Align `rtprio(2)` and POSIX(.1b) interfaces
  - `SCHED_IDLE`
  - Easy to add more
- Linuxulator included
  - `SCHED_BATCH`

# Work in Progress

- Switch RTP_PRIO_NORMAL levels as nice values
    - Implies a nice value per thread
    - SCHED_OTHER?
- Allow kernel threads in the TSP
    - geli
- Priority reporting based on rtprio(2) by default
    - ps(1)
    - top(1)
- Adoption of true SCHED_BATCH
- Fix nice values effect

# Code Status

- Only minor stuff already committed
- 256-queue runqueues and impacts
  - Review series starting at `D45387`
  - Seems to be near completion
- `rtprio(2)` and POSIX implementation revamp
  - Still WIP
  - On GitHub, `OlCe2/freebsd-src`, branch `oc-rtprio_sched`
  - Heads-up for external reviews when matured enough
- Some other WIP not published yet
- Get everything into FreBSD 15, MFC for not too disruptive parts

# Possible Future Work

- Hybrid scheduling
- Alternative schedulers?
- Per-process priority limit
  - Unprivileged users could raise to it
  - May obsolete `mac_priority(4)`
- Runaway processes mitigations
  - Downgrade `SCHED_FIFO` threads to `SCHED_RR` by default
  - Allocate time slots to threads in lower priority classes

## Thanks!

# Questions? Thoughts?

`olce@`

- AsiaBSDCon 2024 paper on `https://papers.freebsd.org/`
- 256-queue runqueues and impacts: Review series `D45387`
- `rtprio(2)` and POSIX implementation revamp: GitHub `OlCe2/freebsd-src`, branch `oc-rtprio_sched`