How to optimize the Linux desktop

An introduction to ulatencyd

https://github.com/poelzi/ulatencyd/

In the perfect world

Userspace behaves nicely

Doesn't leak memory

Doesn't overcommit Ressources

Processes respect more important Tasks

Dream on.

What I expect from a Desktop

- My currently used program must have top priority
- Switching programs must be as fast as possible
- The UI of my desktop should never lag
- No user-space program error should cause a System to crash. UI should still be usable
- Simple user errors/overload should not cause the system to behave badly.
- Scheduling should respect the nature of tasks not processes

What needs to be optimized

- CPU
- IO
- Memory

What needs to be optimized

- Server
  - Resourceless are shares between Services

    Webserver/Database/Fileserver/(Shell)/...
  - Programs don't change
  - Mostly static load

    → static optimization

What needs to be optimized

- Desktop
  - Task oriented
  - Each task can have many processes
  - Important tasks switch constantly
  - Tasks may be unknown
  - Different workloads

    Normal/Games/Videos

    → Highly dynamic workload !!!

    → Requires dynamic optimization

The Scheduler

CPU

- CFS Scheduler (Completely Fair Scheduler)
- O(1) behaviour
- Is completely fair among Tasks of same Priority
- Has different bands

The Scheduler

- Realtime Bands
  - SCHED_FIFO

    a first-in, first-out policy
  - SCHED_RR

    a round-robin policy
- SCHED_OTHER

  the standard round-robin time-sharing policy
- SCHED_BATCH

  for "batch" style execution of processes. CPU bound.
- SCHED_IDLE

  for running **very** low priority background jobs

The ”good” old ages

CPU Nicelevel

- Range between -20 – 19
- -20 most resources / 19 lowest resources
- Non linear behaviour
- Very hard to get determined behaviour
- User can only set 0 - 19

The ”good” old ages

Ulimit

Limits for:

- CPU time
- Open file counters
- Number of process
- Maximum nice level
- Memory usage
- Applies to all child processes

The ”good” old ages

Ulimit

Limits for:

- Applies to all child processes
- Can't be changed afterwards

→ Useless for dynamic changes

Cgroups

- Designed for paravirtualization.
- Filesystem based Interface.
- Allows custom groups of processes
- Allows hierarchies (depending on subsystem)
- Every process is member in exactly 1 group for every subsystem.

Cgroups

Different subsystems for:

- CPU
- Memory
- IO
- *Cpuset*
- *Network*
- *Accounting*
- *...*

Cgroups

CPU:

- **cpu.shares**

  Linear ”percent” of CPU time
- ***cpu.rt_runtime_us***

  micro seconds of cpu time

  for realtime tasks

Cgroups

Memory:

- **memory.limit_in_bytes**

  Limit of physical RAM
- **memory.memsw.limit_in_bytes**

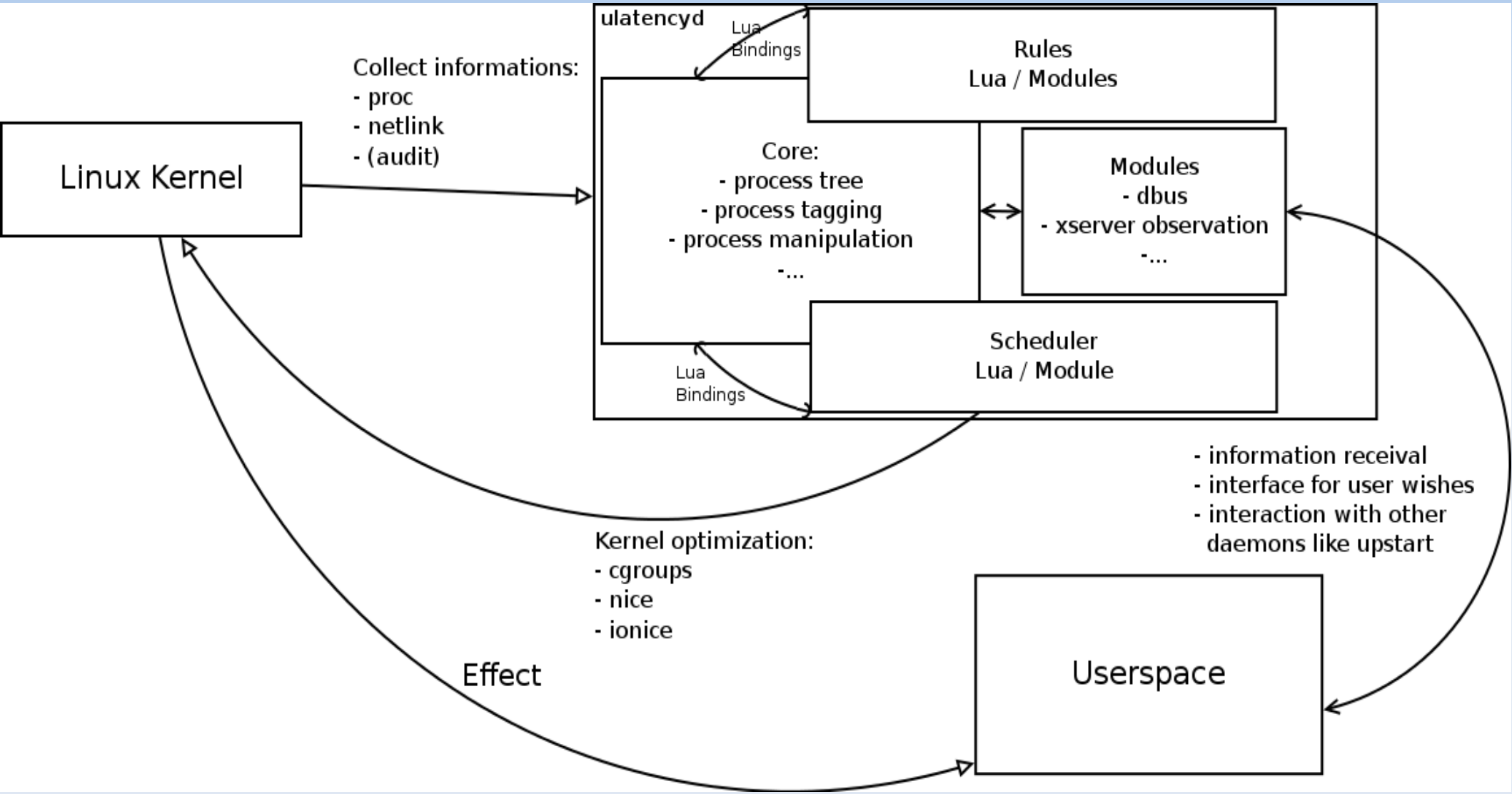
  Maximum limit of total RAM + Swap
- **memory.swappiness**

  Swapiness for a given group

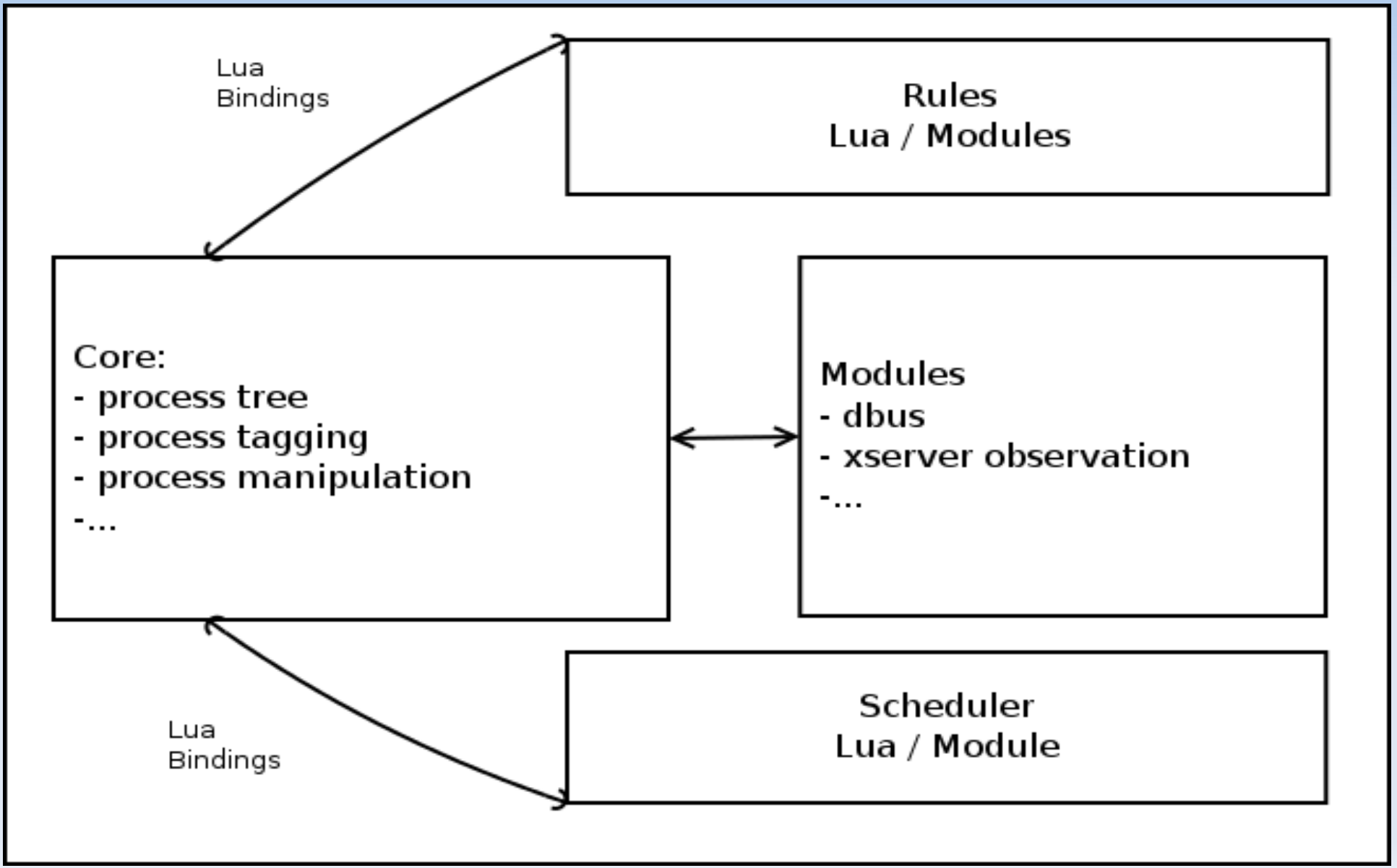
ulatencyd

- Bridge between Kernel Interfaces and Userspace
- Place to workaround userspace bugs
- Highly scriptable
- Easy to extend

ulatencyd


ulatencyd
Lua Bindings
Rules
Lua / Modules
Collect informations:
- proc
- netlink
- (audit)
Linux Kernel
Core:
- process tree
- process tagging
- process manipulation
-...
Modules
- dbus
- xserver observation
-...
Scheduler
Lua / Module
Lua Bindings
- information receival
- interface for user wishes
- interaction with other daemons like upstart
Kernel optimization:
- cgroups
- nice
- ionice
Effect
Userspace

ulatencyd


Lua
Bindings
Rules
Lua / Modules
Core:
- process tree
- process tagging
- process manipulation
-...
Modules
- dbus
- xserver observation
-...
Lua
Bindings
Scheduler
Lua / Module

ulatencyd

- Update process tree

  periodicly/on events
- Run filters

  Set flags on processes
- Run scheduler

  Moves processes between cgroups

ulatencyd

- Process tree
  - ”Objects” in LUA
  - Maps most important /proc/[PID]/* values
  - Easy to use interface for kernel syscalls
  - Pseudo values for read only process values

    session/process group

Filters

- Usually called form the core
- Can register timeout functions

Filters

```
1  DesktopEssential = {
2    name = "DesktopEssential",
3    re_cmdline = "/usr/bin/X",
4    check = function(self, proc)
5      local flag = ulatency.new_flag{name="system.essential"}
6      proc:add_flag(flag)
7      -- adjust the oom score adjust so x server will more likely survive
8      proc:set_oom_score(-400)
9
10     return ulatency.filter_rv(ulatency.FILTER_STOP)
11   end
12 }
13
14 ulatency.register_filter(DesktopEssential)
```

Other Stuff

- D-Bus interface
  - Allows to set/remove flags of users processes
  - Allows to switch configuration
- The active list
  - Stores the last active processes of a user
  - Controllable via DBUS

Plugin: xwatch

- Observes local X11 Servers
- Populates the users active list

Plugin: simplerules

- Handles most simple cases
- Sets flags based on
  - Filename
  - Path
  - Command line
- Example:

```
xfwm4                          user.ui
cmd:python*exaile.py*          user.media
/usr/games/*                   user.game inherit=1
```

The Scheduler

- Decides cgroup for processes
- Sets parameters of groups
- Currently implemented in Lua
- Tree based configuration

The Scheduler

```
-- cpu & memory configuration
SCHEDULER_MAPPING_DESKTOP["cpu"] =
{
  {
    name = "rt_tasks",
    cgroups_name = "rt_tasks",
    param = { ["cpu.shares"]="3048", ["?cpu.rt_runtime_us"] = "949500" },
    check = function(proc)
          local rv = proc.received_rt or check_label({"sched.rt"}, proc) or proc.vm_size == 0
          return rv
        end,
  },
  {
    name = "system_essential",
    cgroups_name = "sys_essential",
    param = { ["cpu.shares"]="3048" },
    label = { "system.essential" }
  },
  {
    name = "user",
    cgroups_name = "usr_${euid}",
    check = function(proc)
              return ( proc.euid > 999 )
            end,
    param = { ["cpu.shares"]="3048",  ["?cpu.rt_runtime_us"] = "100" },
    children = {
      {
        name = "poison",
        param = { ["cpu.shares"]="10" },
        label = { "user.poison" },
        cgroups_name = "psn_${pid}",
      },
```