Suspend Time Compensation For Linux

Baolin Wang from Unisoc (Spreadtrum)

Introduction

Background

Problems

Solution

Future work

Questions

Background

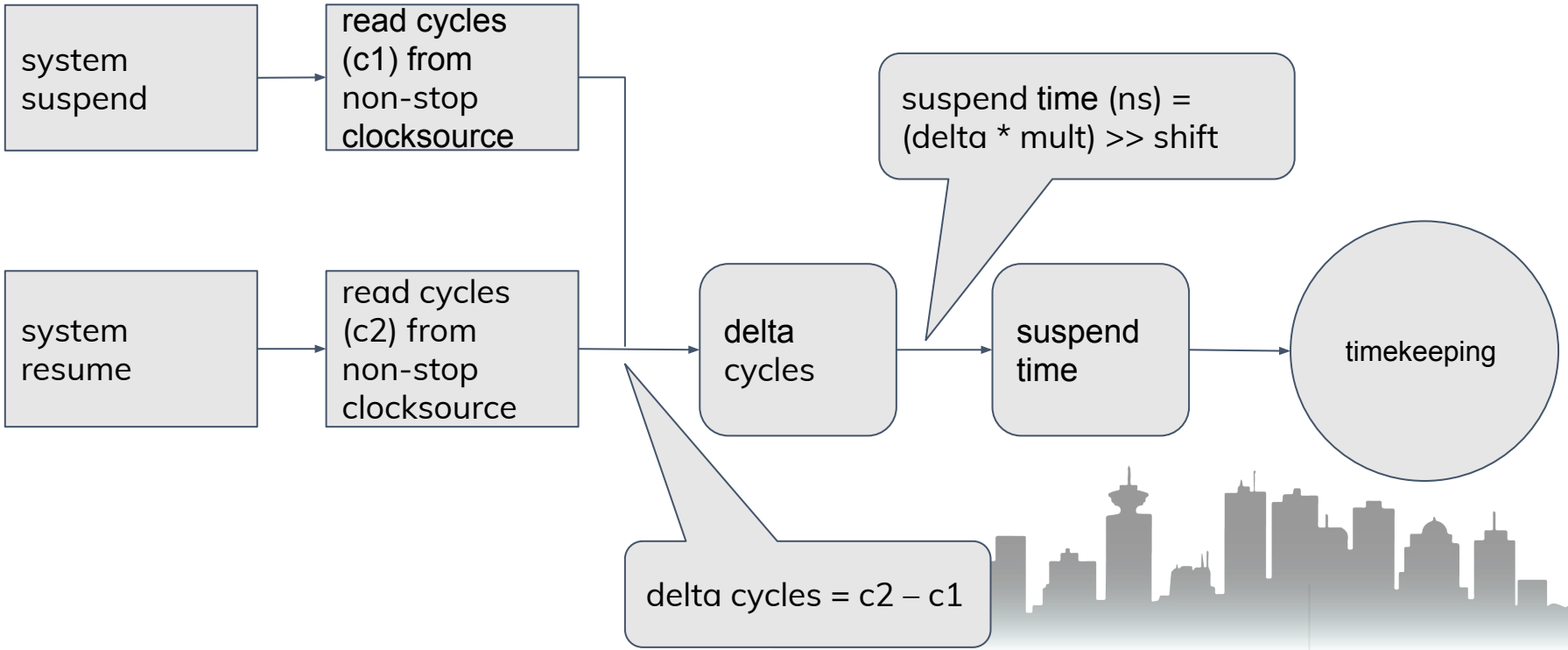
- Why need compensate suspend time
  - The timekeeping will not be updated due to clcoksource/clockevents are suspended
  - System time requirements when system resumes

- There are 3 ways in kernel to compensate the suspend time
  - Non-stop clocksource
  - Persistent clock
  - RTC

- Choose preference
  - Non-stop clocksource ---> Persistent clock ---> RTC

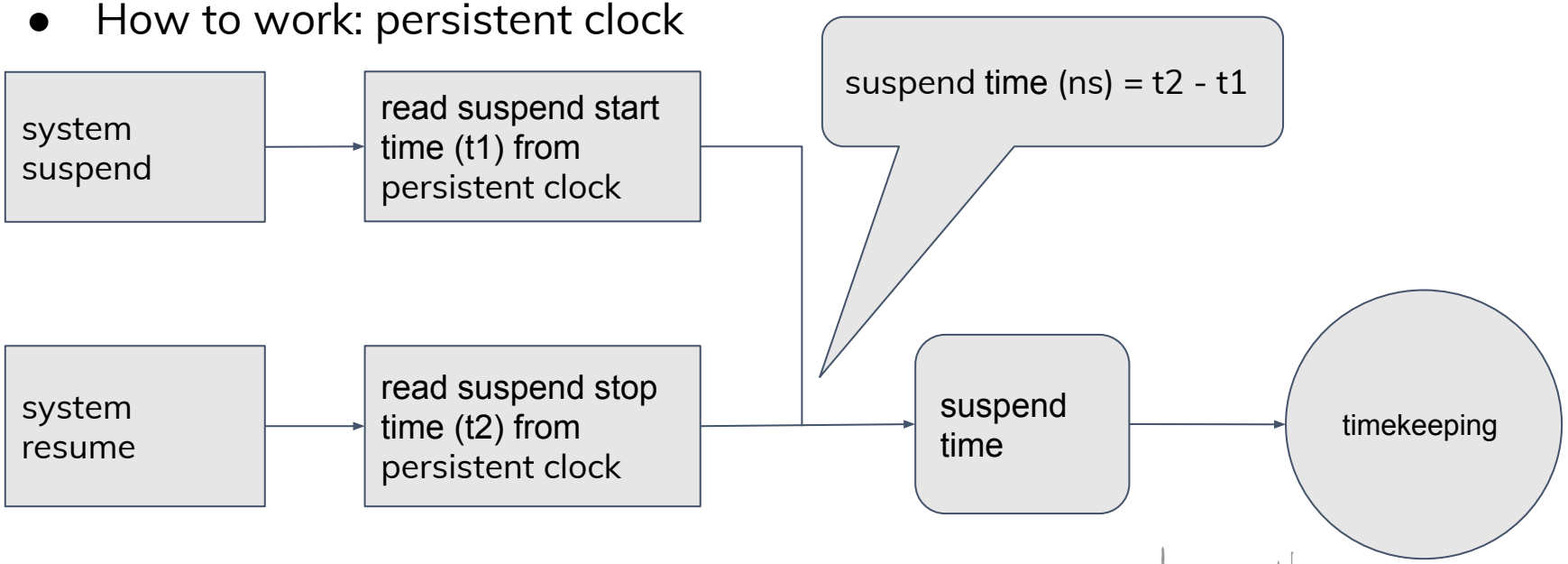
Background

- How to work: non-stop clocksource



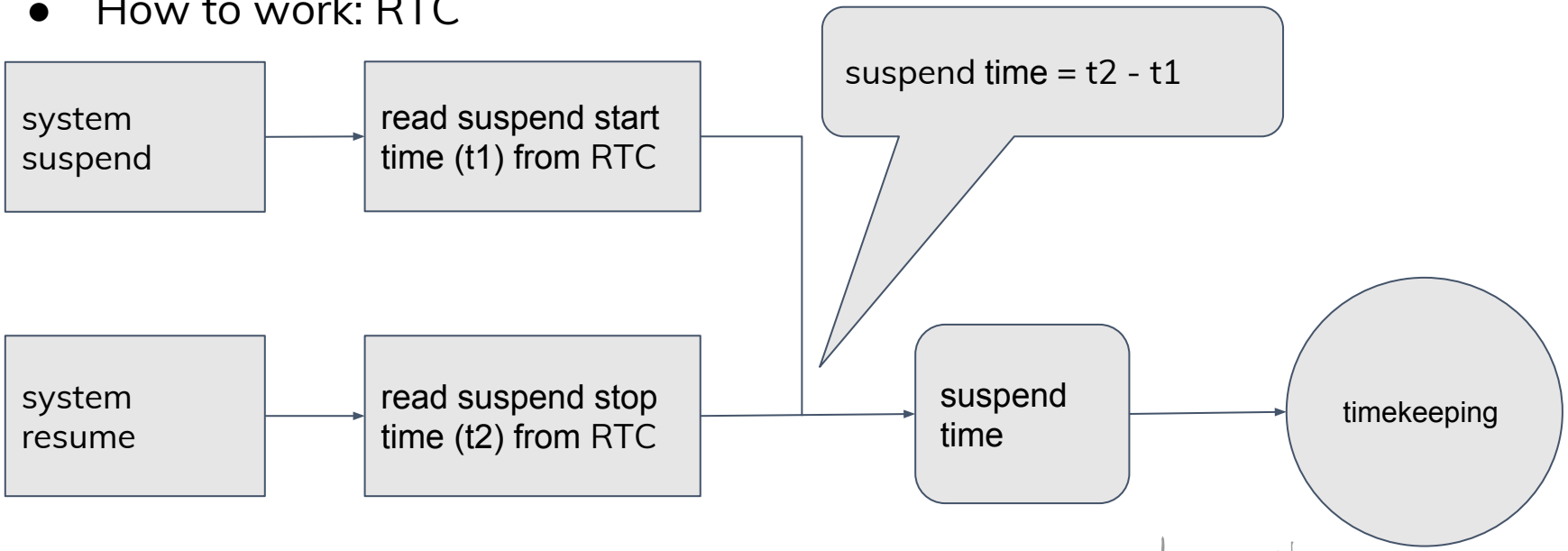
Background

- How to work: persistent clock



Background

- How to work: RTC



Problem for non-stop clocksource

- Now we have logics to compensate the suspend time by non-stop clocksource, but suppose below scenario
  - One high resolution clocksource (cs 1) is selected as the current clocksource for timekeeping, but it will be stopped in suspend
  - Have another low resolution clocksource (cs 2), but non stop in suspend
  - Want to use cs 1 for timekeeping and use cs 2 to compensate the suspend time

- But the timekeeping core only supports the non-stop clocksource if that clocksource is the current clocksource for timekeeping
  - Can not use the low resolution clocksource (cs 2) to compensate the suspend time, should choose persistent clock or RTC instead

Solution1 for non-stop clocksource

- Register one persistent clock using those non-stop timers
  - Implement read_persistent_clock64() in driver

- NAK
  - More duplicate code with calculating extra multiplier/shift and converting cycles
  - Can not be compatible with different architectures
    - Use register_persistent_clock() to register one persistent clock on ARM architecture
    - Implement x86_platform.get_wallclock() on X86 architecture
    - Implement read_persistent_clock64() on ARM64 architecture

Solution2 for non-stop clocksource

- Introduce one common persistent clock framework [1]
  - Each non-stop timer can be registered
  - Supply common code to deal with the mult/shift and cycles converting
  - Implement only one read_persistent_clock64() instead implementing in different architectures

- NAK
  - No reason to invent yet another set of data structures and more read functions with a sequence counter
  - Using mul_u64_u32_shr() can avoid introducing extra mult/shift
  - Clocksources are registered with all required data in the clocksource core, should expand the existing infrastructure to handle that

Solution3 for non-stop clocksource

- Introduce one suspend clocksource for timekeeping/clocksource [2]

- How to select one suspend clocksource
  - Non stop in suspend (CLOCK_SOURCE_SUSPEND_NONSTOP)
  - Do not supply resume/suspend interfaces
  - Pick the best rating
  - Can be current clocksource for timekeeping or not

- Start measuring the suspend timing
  - If current clocksource is the suspend timer, we should use the cycle values from timekeeping as the start-suspend-cycles to avoid same reading from suspend timer
  - If not, enable the suspend clocksource firstly, then read suspend timer cycles as the start-suspend-cycles.

Solution3 for non-stop clocksource

- Stop measuring the suspend timing
  - If current clocksource is the suspend timer, we should use the cycle values from timekeeping as current cycles to avoid same reading from suspend timer
  - If not, read current cycles from suspend clocksource
  - Convert delta cycles to nanoseconds by mul_u64_u32_shr(), inject suspend time for timekeeping
  - Disable the suspend clocksource to save power if possible
- Considering clocksource mutex when calculating the suspend time
  - timekeeping_resume() is called very earlier
  - timekeeping_suspend() is called very late
- Unbind one suspend clocksource
  - Try to install a replacement suspend clocksource
  - If no replacement suspend clocksource, just let the clocksource go and have no suspend clocksource

Solution3 for non-stop clocksource

- Accepted
  - Supply one common mechanism to be compatible with architectures
  - Did not invent new structures which are just copy from clocksource
  - Did not introduce complexity
  - Power saving consideration

Problem for persistent clock

- Weird clock can not be registered as one clocksource or RTC
- Some platforms still implement the obsolete read_persistent_clock()
  - Y2038 issue (struct timespec)
  - Should be replaced with read_persistent_clock64()

Solution for persistent clock

- Remove redundant read_persistent_clock() with removing some redundant architectures (by Arnd Bergmann)
  - /arch/blackfin
  - /arch/frv
  - /arch/m32r
  - ……
- Convert to use RTC instead
  - /arch/sh
- Convert to register as one clocksource
  - /arch/arm/plat-omap/counter_32k.c
- Convert to use read_persistent_clock64()
  - /arch/nios2
  - /arch/parisc
  - /arch/mips
  - /arch/m68k
  - ……

Problem for RTC

- If no non-stop clocksource or persistent clock in system, use RTC to compensate the suspend time
  - Through rtc_suspend/rtc_resume interfaces

- Y2038 issue and expiration risk
  - 1 driver will be expired before year 2017
  - 7 drivers will be expired before year 2038
  - 23 drivers will be expired before year 2069
  - 72 drivers will be expired before 2100
  - 104 drivers will be expired before 2106

Solution for RTC

- Introduce RTC hardware ranges [3]
  - Add fields ‘range_max’ and ‘range_min’ for RTC
  - Drivers can set the ranges
  - Valid ranges in RTC core instead of each driver

- Add one RTC range offset to expand RTC range [4]
  - Add one field named ‘start_secs’ for RTC
  - Can be set through DT or drivers
  - Unit is seconds
  - RTC core will calculate the offset seconds between the ‘start_secs’ and RTC hardware range
  - Add the offset to the time when reading from hardware, and subtract it when writing back
  - Avoid leap year issue

Future work

- Persistent clock
  - Remove obsolete read_persistent_clock()
  - Convert to RTC driver or clocksource instead
- RTC
  - Fix Y2038 issue
  - Add hardware ranges for each driver
- Still there are 3 cases, can be simplified?

Questions

- Q & A

Thanks

Thanks Arnd Bergmann, John Stultz, Mark Brown, Daniel Lezcano

[1] https://patchwork.kernel.org/patch/10397555/
[2] https://git.kernel.org/torvalds/c/39232ed5a179
https://git.kernel.org/torvalds/c/156955754969
[3] https://git.kernel.org/torvalds/c/71db049e7355
[4] https://git.kernel.org/torvalds/c/989515647e78