We Are Not Getting Any Younger

(A New Approach to Timekeeping and Timers)

Nishanth Aravamudan, Darren Hart,
and John Stultz
IBM Linux Technology Center

About Me

- I'm John Stultz (johnstul@us.ibm.com)
- Work for the LTC xSeries Kernel Team at IBM
  - Translation: i386, x86_64, ia64

What I'm Going to Cover

- How Linux currently keeps time and some of its problems
- Proposed design for timekeeping
- Future extensions

Expectations of `gettimeofday()`

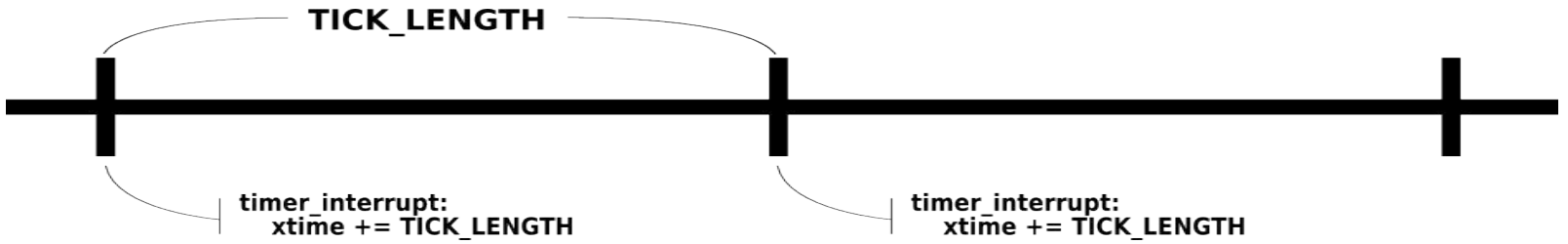
- Correctness
  - Time should NEVER go backwards
- Should accurately keep track of time
  - No jumps or plateaus in the flow of time
  - Time should never be lost
- Finely grained micro-second or better resolution
- Adjustable for hardware drift

Customer Issues

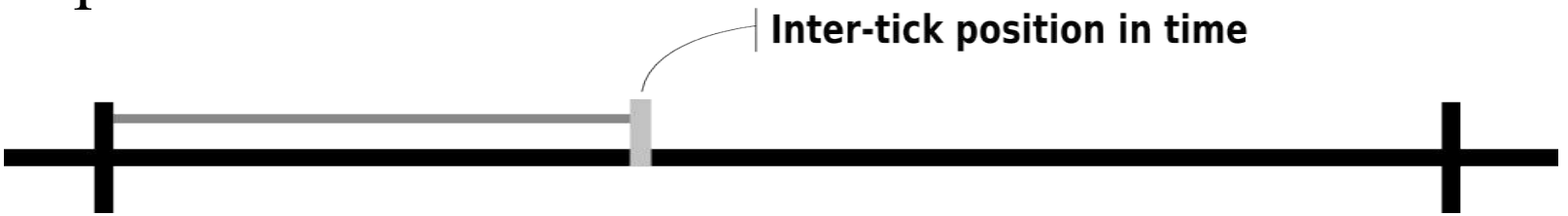
- Using gettimeofday() for time-stamping transactions, logging, and performance analysis
  - Do not tolerate any time inconsistencies
- Want very fast and very finely grained time
  - Performance is a big issue
- Desire close time synchronization between systems for distributed applications

How Timekeeping Works

- Tick based
  - Every tick, we increment `xtime` by one tick interval

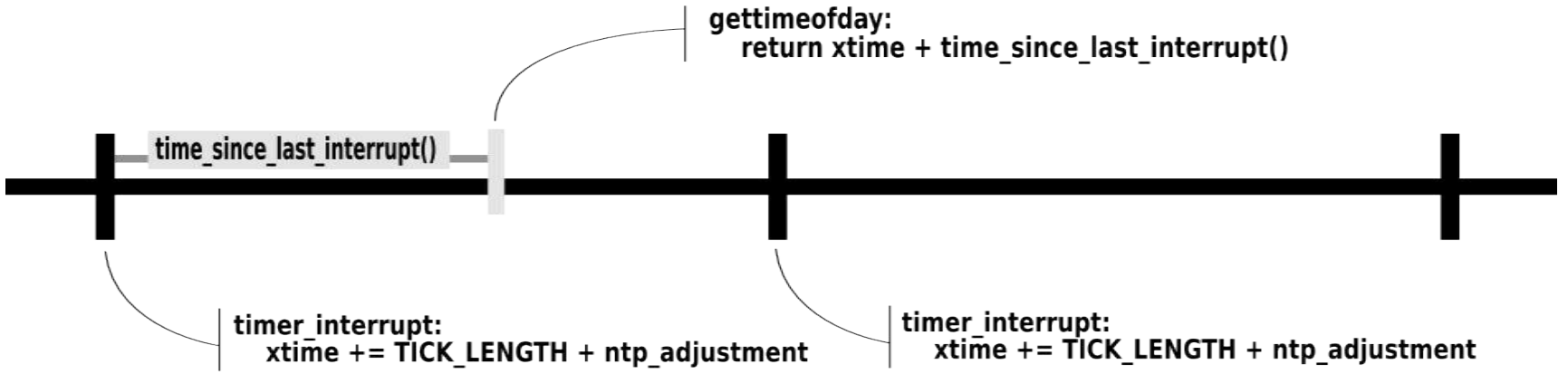


- Use Interpolation
  - Use a high-res timesource to calculate the inter-tick position in time



Pseudo-Code

```
timer_interrupt:
   delta = TICK_LENGTH + ntp_adjustment
   xtime += delta
gettimeofday:
   delta = time_since_last_interrupt()
   return (xtime + delta)
```



Problematic Design Issues

- Assumption that ticks are never late or lost
- Inconsistently measuring time
  - Inconsistent use of NTP adjustment
  - In `timer_interrupt()` we accumulate time differently then how it is calculated in `gettimeofday()`
- Interpolating between two time sources
  - When something goes wrong, which one is right?

How Problems Can Occur

- Late or missed ticks
  - Drivers disabling interrupts (IDE PIO)
  - Timer interrupt starvation (APIC mode)
  - BIOS SMIs can take up to 30-50ms
  - Virtualization (Xen)
- Poor calibration between interrupt source and high-res time source
- NTP adjustments not made consistently

Developer Confusion

- Lack of a clear internal interface
  - Just a bunch of global variables
  - Terrible variable names
- Poorly documented variable dependencies
  - Changing `xtime` requires changing `wall_to_monotonic`
  - NTP interactions
- Misunderstood variable independencies
  - `jiffies * HZ != xtime + wall_to_monotonic`

Maintenance Headaches

- Increasing number of architectures
- Shared hardware components
- Lots of code reimplementing the same basic thing.
- **`grep -i "this is revolting" -r *`**

Proposal: Goals

- Consistency
  - Increment timekeeping variables without interpolation
    - Using same method for `gettimefoday()`
  - Consistent use of NTP adjustments
- Isolation
  - Split timer subsystem from timekeeping
  - Black-box NTP state machine
- Reduce Duplication
  - Generic timekeeping algorithm
  - Hardware specific timesources

Proposal: Overview

- NTP changes
  - Isolate and cleanup in kernel NTP state machine
- Timesources
  - Introduce a flexible driver-like abstraction of a free running counter
- New timekeeping algorithm
  - Consistently use timesources and NTP adjustments to avoid inconsistencies

Proposal: NTP

- Isolate all NTP code into ntp.c
- Make NTP variables static
- Create clean and clear interfaces
  - `ntp_adjtimex()`
  - `ntp_advance()`
  - `ntp_clear()`
  - `ntp_synced()`

Proposal: Timesources

- Abstraction of a free running counter of known frequency
- Can be treated as a driver
  - Arches with similar hardware can use the same timesource driver
  - Can be loaded while the system is running
  - Multiple timesources can be available

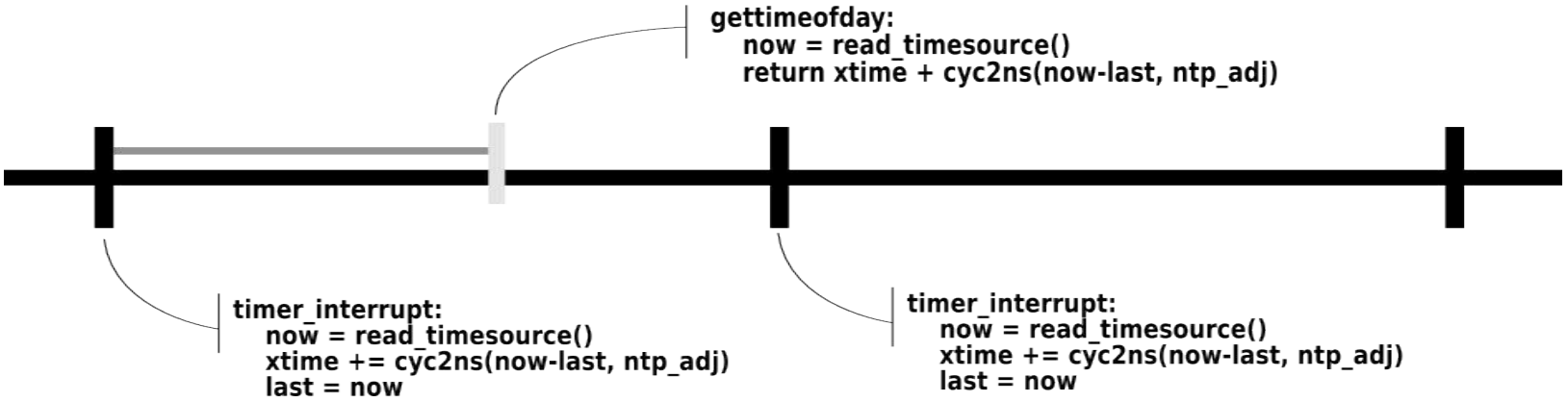
Proposal: Core Algorithm

- Utilizing the two previous improvements in an arch generic fashion
- Consistent NTP adjustments
  - Don't add or subtract time at every tick
  - Make adjustments to the timesource frequency
- Consistently use the timesources
  - Accumulate time in the same way we calculate it in `gettimeofday()`

Core Algorithm: Pseudo Code

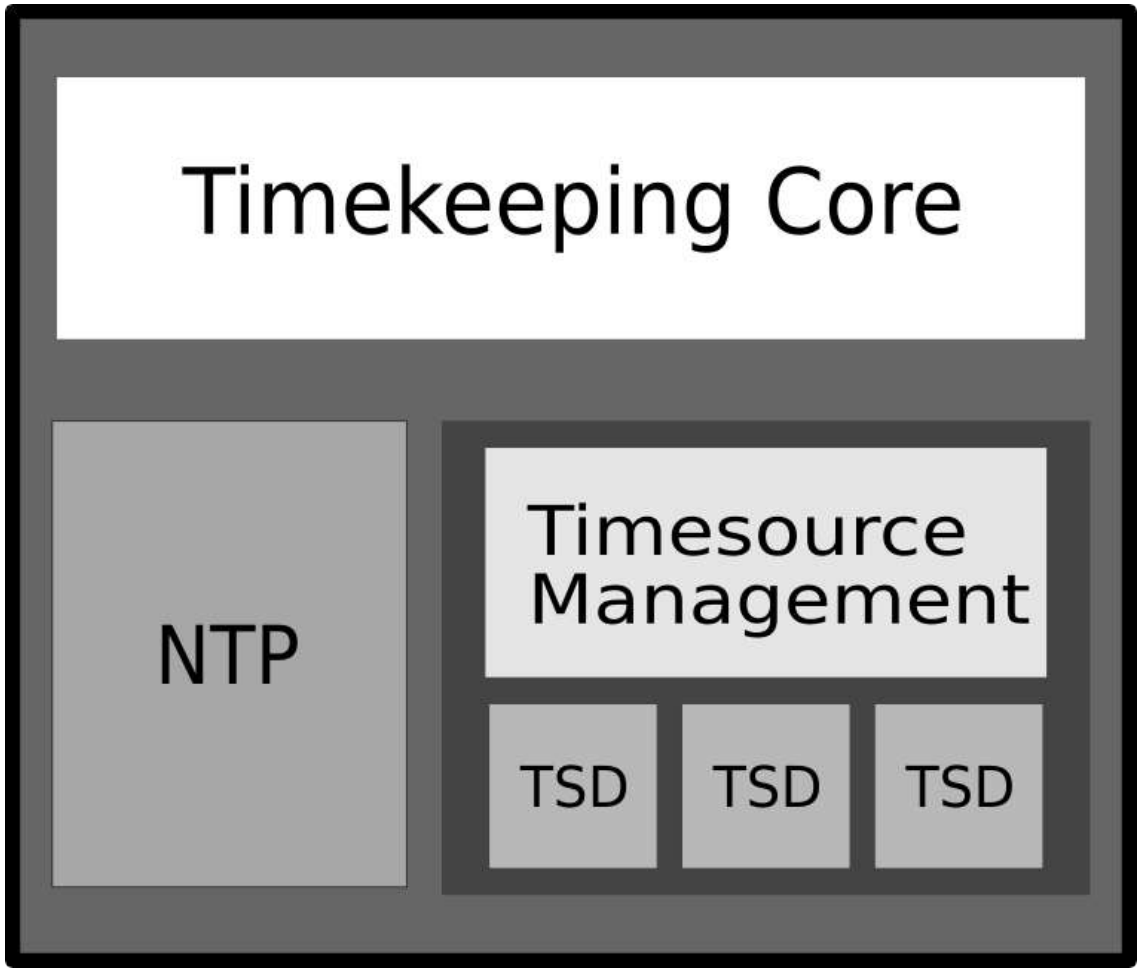
```
timer_interrupt:
   now = read_timesource()
   delta = cycles_to_ns(now - last, ntp_adj)
   xtime += delta
   ntp_adj = ntp_advance(delta)
   last = now

gettimeofday:
   now = read_timesource()
   delta = cycles_to_ns(now - last, ntp_adj)
   return (xtime + delta)
```

Core Algorithm: Details



- No assumptions about tick length!
  - Late or lost ticks do not affect timekeeping
- We can do the periodic timekeeping outside of `timer_interrupt()`
  - Use soft-timer to reduce interrupt overhead
  - Run every so many milliseconds (currently 50)

Proposal: Block Diagram



Exposed Interfaces:

```
do_gettimeofday()

do_settimeofday()

do_adjtimex()

do_monotonic_clock()

timer_interrupt_hook()
```

Proposal: Concerns

- Large change to a delicate subsystem
- Use 64 bits of nanoseconds as the base time unit
  - Embedded folks don't like this
  - Worse case, fall back to using a timespec
- Some systems do not have free running counter
  - Origin of interrupt/offset timekeeping
  - Interpolation is allowed internally in the timesource driver

Proposal: Status

- Core code complete
- Basically finished on i386, x86_64
- Functional on ppc, ppc64
- Started work on alpha, ia64, s390, sparc, sparc64
- Sent to -mm for inclusion
  - OH NO! Patch too big to discuss well!
- Currently breaking it up into smaller bits to address concerns individually

Proposal: Summary

- Insures correctness via consistent usage of timekeeping variables
- Separates timekeeping from timer subsystem
- Consolidates a large amount of duplicate code
- Cleans up a neglected subsystem
- Paves the way for future improvements that would otherwise be tangled with timekeeping

Future Enhancements

- `gettimeofday()` performance improvements via vsyscall `gettimeofday()` implementations
- Ideas for a time based instead of tick based soft-timer subsystem
- Various variable frequency tick projects
  - _NO_IDLE_HZ, Dynamic Ticks, VST
- Hopefully will simplify inclusion of other works

vsyscall `gettimeofday()`

- Method to avoid syscall overhead.
- Allows for user-accessible kernel pages that include data and code for `gettimeofday()`
- Already implemented in x86_64
- Supported by my proposal
- With my proposal, it was easy to implement for i386

Time Based Soft-timers

- Changes soft-timers to based on time instead of ticks
  - Intuitive and accurate units
  - Avoids latency added from rounding time to ticks
  - Avoids accumulated latency caused by lost ticks
- Allows for flexible interval granularity
  - Tick period could be dynamically adjusted
  - Adapts well to tick-less and virtualized systems
- Better statistics

Detail: Time Based Soft-timers

- Minimally invasive
  - Small patch
  - Supports existing interfaces
  - Modification to existing algorithm
- Actually improves latencies!
  - Best case is **much** closer to requested
  - Average case is 1.5 times tick frequency

Variable Frequency Tick

- Lower system overhead by disabling timer interrupts when there are no soft-timers to expire soon.
  - Improves power saving and performance.
  - Allows virtualized systems to let other partitions run longer
- Different Approaches
  - NO_IDLE_HZ
  - Dynamic Tick
  - Lots of others.

Detail: Variable Frequency Tick

- Problem: It is difficult to keep proper time if timer ticks are disabled
- My proposal allows for timer ticks to be skipped without affecting timekeeping
- Specific implementations can reprogram the interrupt source without worrying about timekeeping.

High-Res Timers Help?

- Customers are demanding High-Res Timers
- Can we reduce the impact HRT has to the kernel?
- Allow for a more generic implementation?
- Working with HRT maintainers to address these issues.
- Lots of good discussion at the informal BOF

Summary

- Went over how timekeeping currently works
  - Discussed some of the design problems
- Covered my proposal for a new timekeeping subsystem.
- Looked at some future extensions to the Linux kernel that my proposal helps enable.

Legal Statement

This work represents the view of the author and does not necessarily represent the view of IBM.

Questions?

Timesources: Detail

```
struct timesource_t {
    char* name;
    int priority;
    enum {
        TIMESOURCE_FUNCTION,
        TIMESOURCE_CYCLES,
        TIMESOURCE_MMIO_32,
        TIMESOURCE_MMIO_64
    } type;
    cycle_t (*read_fnct)(void);
    void __iomem *mmio_ptr;
    cycle_t mask;
    u32 mult;
    u32 shift;
    void (*update_callback)(void);
};
```

Variable frequency ticks

- Changing timer interrupt freq while system is running (not looking ahead)
  - Linus' skip 1, 2 or 4 idea?
  - Incrementing jiffies based on time?
  - PPC64 like time based soft-ticks?
- Looking ahead to change frequency when idle
  - Need to consider cost of soft-timer list structure
- Both are still inter-tick unaware, so we have to be careful with add/mod_timer

High Res Timers

- Single list, or dual mode lists?
  - How are we storing and migrating these?
- Bind timer_base lists to interrupt sources (instead of cpus)?
  - HRT does add_timer_on(base_hpet,...)
- Export arch specific hardware to posix interface?