Timers in Linux

- Backing wall clock time
  - As reliable as possible
  - Regular timer interrupt
- Offset timer
  - Offset from last wall clock tick
  - gettimeofday
- Per CPU interval timer
  - Needed on MP and everything will be MP ...
  - Used for most events and scheduling
  - Local APIC, but difficulty with C3.
  - Access/Change should be fast for reprogramming

All x86 timers are broken in different ways

- PIT
  - Slow
  - Awkward programming
  - Max timeout too small
- HPET
  - Global
  - Most BIOS just don’t allow it or broken
  - Access still slow
- PM/ACPI Timer
  - Slow
  - Sometimes broken
  - Small timeout
- TSC
  - Fast
  - Not synchronized on some systems
  - No interrupt
- Local APIC timer
  - Per CPU
  - Fast
  - But stops ticking on some systems in C3 :-(

Towards variable interval timers

- Increasing frequency
  - Costs CPU (~5% 100->1000)
  - Causes lost ticks with slow drivers or SMM
- Variable for powersaving and virtualization and faster timeouts
- Requirements
  - Per CPU
  - Reprogramming/Recovering time fast and without drift
- None of the existing timers suited well

gettimeofday/clock_gettime

- Most frequent syscall by far
  - Critical for some workloads like databases
- Requirements
  - Monotonic over all CPUs
  - Accurate
  - Fast
  - Should be securely accessible in ring 3 code
- TSC in theory great
  - But needs to be synchronized over systems
- Fallback is painful
  - External timer very slow (factor 3-4, getting worse)

Time accounting

- Periodic interrupt to sample sys/user/intr
  - Large sampling error
  - Costs CPU and limits virtualization
- Microstate accounting
  - RDTSC on each kernel/entry exit
  - Impacts fast paths

Performance counters

- Basic use cases:
  - System global (oprofile,vtune)
  - NMI watchdog
  - per process virtualized (not yet mainline)
  - hypervisor per guest
- Most needs relatively simple
- Only using a small subset of basic counters
  - Standard CPU time (for kernel too)
  - TLB miss
  - Cache/TLB misses