IRQ next prediction

Daniel Lezcano
Introduction

“Idle” opposed to “Running”, in between an event occurred

- We want a cpu to be idle as much as possible without impacting the performance of the system
- The cpuidle framework takes care of entering the idle state depending on the predictable events on the system
- The depth of the sleep will depend on the next event on the system and the latency constraints
- An event makes a CPU to exit its idle state
- Some lecture:
  - Computing a target residency: https://goo.gl/aywVSl
  - Sources of wake up: https://goo.gl/zXpQfW
The idle task

- The idle task is the last task executed on the CPU
- The idle task is an infinite loop
  - Scheduled out when ‘need_resched’ is set
  - Otherwise enters the idle function
Sources of wake up

● Three kinds of interruption:
  - Inter processor interrupts (IPI)
  - I/O interrupts
  - Timer interrupts
Inter Processor Interrupt

- Used conveniently to inform or remotely execute an action on another processor

- Very few relevant:
  - Timer broadcasting: used to notify another CPU a timer expired. Pointless if there is an always-on per cpu timer or negligible if the FEAT_DYNIRQ is set for the timer
  
  - Rescheduling interrupt: the most important IPI, basically used to wake up an idle CPU when there is a task in the runqueue, mastered by the scheduler.
    - Pitfall: it could be a wakeup resulting from an IO completion where the interrupt was received on another CPU
Timer

- **Per cpu timer:**
  - a private irq line

- **Global timer:**
  - Can wakeup any CPUs randomly but:
    - in practical the DYNIRQ feat will make it to expire on the CPU with the earliest timer
    - the cpumask is usually set to CPU0

- The next event is predictable on this device
I/O interrupt

- Some are coming from devices with internals communication:
  - MMC, I2C, SATA, GPU, …
  - May be they have repeating patterns?

- And others from external events:
  - Keyboard, network, mouse, …
  - May be they can have repeating patterns?

- Intuitively the former can have a burst of interrupt activity followed by long quiescent point, the latter can be considered as slow devices and a prediction failure has a negligible impact
Current approach: the glitch

- Measures how long it has been in the idle function
  - Statistics are done on the idle durations

- Reminder: a CPU is woken up by:
  - Timers, IPI and device interrupts

- So the statistics are trying to predict the next interrupt but that includes:
  - The next timer expiration !?
  - The next scheduler decision !?
Current approach: the soup

- The governors are mixing different kind of information
  - Deterministic and non deterministic

- It is impossible to identify the source of the wake up and:
  - Discard some events like the timer ones
  - Identify if the observed interrupt is the expected one
  - Create a feedback loop prediction vs observation
Let’s analyze the devices

- **Recipe:**
  - Using well known benchmarks
  - Pin the observed interrupt on a specific CPU
  - Used trace-cmd with the sequence ‘start’, <cmd>, ‘stop’, ‘extract’ in order to prevent writes to the disk during the test
  - For the observed interrupt, measure the interval between the occurrences and their frequencies
    - Intervals representation gives the chronology of the events and the repeating patterns
    - Frequencies give the probabilities
      - A peak means is a high probability
MMC / SD card

- Writing on the SD card with sysbench test
  - `sysbench --test=fileio --file-test-mode=seqrewr --file-total-size=128M run`
SATA / SDD

- Writing on the SSD 6Gb/s with sysbench test
  - `sysbench --test=fileio --file-test-mode=seqrewr --file-total-size=128M run`
WiFi

- Copying a file with WiFi from Internet
  - ketchup v4.10
Ethernet / Gb

- Copying a file 1.8GB with a 1Gb/s ethernet on a local network via NFS
  - `cp /net/shared/myfile /tmp`
Console / UART

- Writing to the console via UART
  - dmesg
SATA / HDD

- Writing on a HDD 3Gb/s with sysbench test
  - `sysbench --test=fileio --file-test-mode=seqrewr --file-total-size=128M run`
Observation

- The frequencies show the peaks
  - Several peaks => there is a pattern
  - A single peak => averaging and discarding anomalies is enough

- The intervals are stable, we can see a pattern for some devices
  - Visible with kernelshark and on the graphics
Hypothesis

● If we can find the pattern of the interrupts, then we can predict the next event more accurately for each interrupt
  - What algorithm?

● Single peak can be computed with a simple average on the last 32 samples (rule of thumb) assuming it is a normal law

● Taking the earliest next events from all interrupts gives the next event
Challenges

- A pattern rather than a periodic interval could be hard to predict

- The implementation is full of pitfalls
  - Missing information (IPI IO, per cpu timer flag, etc ...)
  - Mathematic with large numbers
  - The idle place is very special (lock, rcu, interrupt, ...)

- Statistics computation consumes energy and takes time
  - What is the best trade-off?

- New tools to do an instrumentation (prediction accuracy, spot where the problem is, ...)
Status

TBD

- Video
- Audio
- Idle
- User Interface
- Browsing
What’s next?

- Change the scheduler to be more energy aggressive
- Prevent to wake up an idle CPU which did not reach its target residency
- IPI are also used to wake up a CPU after an IO completion
  - How to discriminate?
  - Change the blocked task on IO wake up path