Soft timers: efficient microsecond timer support for network processing

Mohit Aron
Peter Druschel
Background and Motivation

• CPUs rely heavily on pipelining and caching
• Interrupts and context switches are very expensive
  – Saving and restoring CPU states; shift in locality causing TLB and cache misses
• Frequency of interrupts becomes an important criterion
Why are interrupts expensive?

- Cause a shift in locality
- Saving and restoring of CPU states is required
- Interrupts are expensive if they occur too often
- Processing can begin only after the thread has been serviced
Conventional timers

- Conventional timers schedule events by invoking a designated handler in context of a h/w interrupt
  - Hardware interrupts given highest priority
  - Cache and TLB pollution due to context switch
  - Cost unacceptable in a fast webserver
Why are s/w timers possible?

- Event handlers can be invoked at certain trigger states under normal operation
- Timer events can be stored and invoked when the system reaches any one of the trigger states
- Events are scheduled in the order in which they arrive (timing wheels can be used to store these events)
Trigger states

- At the end of executing a system call
- At the end of an interrupt handler
- CPU in an idle loop
- At the end of executing an exception handler
  - E.g. Page faults, arithmetic exception
Advantages of software timers

- Allow scheduling of events at a low granularity
- Are useful in scheduling events in a highly loaded system
- Rate based clocking etc. benefit from such facility
- Performance scales with CPU speed
- Are cache friendly
- Easily integrated in the current systems
Applications of software timers

• Rate based clocking in TCP
  – Transmission of packets independent of ACK arrival rate.

• Network polling.
  – Polling the network interface for arriving packets or completion of transmission.
Rate based clocking in TCP

- Conventional TCP depends on ACKs to pace transmission
- ACK compression causes the events to lose temporal spacing
- Bursty ACKs cause a burst of packets transmitted causing congestion
- Slow start solves the temporal spacing but underutilizes the network bandwidth
Conventional solutions

- Transmission in TCP independent of the ACKs
- Periodic hardware interrupt scheduled at intended rate of transmission
- Ideally one packet is transmitted per timer interval
- Fails at higher transmission rates as the context switch becomes significant
Rate based clocking

- Transmission rate is variable due to probabilistic nature of the timer
- Two parameters:
  - Target transmission rate
  - Maximal allowable burst
- Algorithm keeps track of transmission rate
- Next transmission scheduled for meeting target transmission rate, bursts allowed in case of delays
Network Polling

• NI generates h/w interrupt to indicate completion of transmission
• Upon receiver interrupt, system accepts a packet, performs processing and wakeups
• CPU can poll at specified intervals and avoids interrupts and cache pollution
• Aggregation of packet processing possible in polling
Soft timer implementation

- Schedules events at one of the trigger states
- As a back up to the trigger states there is a hardware timer interrupt which forces the system to enter a trigger state
- Events are stored in order of arrival and dispatched in order
Implementation details

• Supports following operations
  – measure_resolution()
    • 64 bit clock resolution
  – measure_time()
    • 64 bit value returning current time
  – schedule_soft_event( T, handler )
    • schedule handler at least T ticks in future
  – interrupt_clock_resolution()
    • The expected minimal resolution of the clock
Useful range of soft timers

- Useful for events that
  - need rescheduling at very short intervals
  - can tolerate delays up to the hardware interrupt granularity

- Event fires within following bounds
  \[ T < \text{Actual event time} < T + X + 1 \]
  \[ T = \text{Actual time} \]
  \[ X = \text{Interrupt clock resolution relative to the measurement clock} \]
Metrics for soft timers

- Overhead of calling the appropriate event handler
- Event granularity
- Trigger interval distribution over time
- Trigger interval distribution by event source
Base overhead for h/w timer

- NULL handler invoked at a periodic hardware interrupt on an already saturated system
- Interrupt frequency varied as the frequency of the hardware timer
- The base overhead thus found is around 4.36 to 8.64 µsecs
Base overhead for soft timers

- TLB misses not used as a trigger state
- Additional trigger states introduced in system calls
- Handler invoked at every trigger i.e. At the maximal frequency
- Handler called every 30 μsecs
- No impact on the webserver throughput even on high loads
Trigger interval distribution

• Across a variety of applications the trigger intervals were from 2 μsec to 1ms (which is the interrupt clock)
• On a faster CPU the distribution is similar with a reduction in intervals proportional to the clock speed ratios
• There is not much variability due to context switches
Rate based clocking with soft timers

• Throughput increases by around 35%
• Overhead of soft timers is minimal
• Transmission interval is higher in case of Apache and lower for Flash
  – Apache does not take advantage of locality
• Rate of transmission very close to target rate
Network polling with soft timers

- Adaptively set polling interval to find a given number of packet per poll interval on an average
- Throughput increases by 3% to 25% on given workload
- Useless processing in case of lightly loaded system avoided by turning polling off
- Aggregation of packets is possible in this scheme
Conclusion

- Soft timers allow us to schedule events at the granularity of 10s of µsecs with low overhead
- The useful range is bounded by the hardware timer clock
- Scale with the CPU speed and are cache friendly
- Easily integrated with current systems
- Cannot however be used with real time systems
Figure 1. Lower and upper bounds for event scheduling
Figure 6. Impact of event sources on trigger interval, CDF (ST-Apache workload)
<table>
<thead>
<tr>
<th></th>
<th>Apache</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Throughput (conn/s)</td>
<td>774</td>
<td>1303</td>
</tr>
<tr>
<td>HW timer throughput (conn/s)</td>
<td>560</td>
<td>827</td>
</tr>
<tr>
<td>HW timer Ovhd (%)</td>
<td>28</td>
<td>36</td>
</tr>
<tr>
<td>HW timer Avg xmit intvl (µsecs)</td>
<td>31</td>
<td>35</td>
</tr>
<tr>
<td>Soft timer throughput (conn/s)</td>
<td>756</td>
<td>1224</td>
</tr>
<tr>
<td>Soft timer Ovhd (%)</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Soft timer Avg xmit intvl (µsecs)</td>
<td>34</td>
<td>24</td>
</tr>
</tbody>
</table>

*Table 3.* Overhead of rate-based clocking