Real-Time Performance of Linux


OS Latency

**Definition [OS Latency]**

Let $T$ be a task belonging to a time-sensitive application that requires execution at time $t$, and let $t'$ be the time at which $T$ is actually scheduled; we define the OS latency experienced by $T$ as $L = t' - t$. 
Sources of OS Latency

- **Timer Resolution** \((L_{\text{timer}})\)
  - Timer are generally implemented using a periodic tick interrupt. A task that sleeps for an arbitrary amount of time can experience some timer resolution latency if its expected activation time is not on a tick boundary.

- **Scheduling Jitter** \((L_{SJ})\)
  - Task is not highest in scheduling queue.

- **Non-Preemptable Portions** \((L_{NP})\)
  - Latency can be caused by non-preemptable sections in kernel and in drivers. (e.g. ISRs, bottom halves, tasklets).

Timer Resolution

- Standard Linux timers are triggered by a periodic tick interrupt.
- On x86 machines it is generated by the Programmable Interval Timer (PIT) with period \(T_{\text{tick}} = 10\)ms.

- How about decreasing \(T_{\text{tick}}\)?

- High-resolution timers using aperiodic interrupt capabilities in modern APICs (Advanced Programmable Interrupt Controller).

- Timer resolution possible in range of 4-6musec.
Non-Preemptable Section Latency

- **Standard Linux:**
  - Monolithic structure of kernel.
  - Allows execution of at most one thread in kernel. This is achieved by disabling preemption when an execution flow enters the kernel, i.e., when an interrupt fires or when a system call is invoked.
  - Latency can be as large as 28ms.

- **Low-Latency Linux:**
  - Insert explicit preemption points (re-scheduling points) inside the kernel.
  - Implemented in RED Linux and Andrew Morton’s low-latency patch.

- **Preemptable Linux:**
  - To support full kernel preemptability, kernel data must be explicitly protected using mutexes or spinlocks.
  - Linux preemptable-kernel patch disables preemption only when spinlock is held.
  - Latency determined by max. amount of time for which a spinlock is held plus maximum time taken by ISRs, bottom halves, and tasklets.

- **Preemptable Lock-Breaking Linux:**
  - Spinlocks are broken by releasing spinlocks at strategic points.

Preemptable Lock Breaking: Example

```c
void preempted_kernel() {
    spin_lock();
    // Critical section...
    spin_unlock();
}
```

- This function reclaims cached dentry structures in `fs/dcache.c`.
- High-latency point.
- Why count iterations at all?
Test Programs

- Measuring $L_{\text{timer}}$:
  - Run test task on lightly loaded system, to avoid $L_{np}$.
  - Set up a periodic signal (using `itimer()`)

- Measuring $L_{np}$:
  - Run test task against background tasks
  - Test Task:
    - Read current time $t_1$
    - Sleep for a time $T$
    - Read time $t_2$, and compute $L_{np} = t_2 - (t_1 + T)$
  - How to read $t_1$ and $t_2$? (`gettimeofday()`?)

Timer Latency

Figure 1. Inter-Activation times for a task that is woken up by a periodic signal with period $100\mu$s on a high resolution timer Linux.

Figure 2. PDF of the difference between inter-activation times and period, when $T = 1000\mu$s.
Test Programs

- Measuring $L_{timer}$:
  - Run test task on lightly loaded system, to avoid $L_{np}$.
  - Set up a periodic signal (using `itimer()`)

- Measuring $L_{np}$:
  - Run test task against background tasks
  - Test Task:
    - Read current time $t_1$
    - Sleep for a time $T$
    - Read time $t_2$, and compute $L_{np} = t_2 - (t_1 + T)$

  - How to read $t_1$ and $t_2$? (`gettimeofday()` ?)

Measuring $L_{np}$

- **Memory Stress:**
  - Page fault handler invoked repeatedly.

- **Console-Switch Stress:**
  - Console driver contains long non-preemptable paths.

- **I/O Stress:**
  - Systems calls that move large amounts of data between user and kernel space, or from kernel memory to hardware peripherals.

- **Procs Stress:**
  - Concurrent access to `/proc` file system must be protected by non-preemptable sections.

- **Fork Stress:**
  - New processes created inside non-preemptable section and requires copying of large amounts of data.
  - Overhead of scheduler increases as number of active processes increases.
OS Non-Preemptable Section Latency

Figure 3. OS non-preemptable section latency measured on a high-resolution timer Linux. This test is performed with heavy background load.

Background Load Tests

Standard Linux
Background Load Tests

Low-Latency Kernel

Preemptable Kernel
Background Load Tests

Lock-Breaking Preemptable Kernel

OS Non-Preemptable Portion Latency

<table>
<thead>
<tr>
<th></th>
<th>Memory Stress</th>
<th>Caps-Lock</th>
<th>Caps-Lock Stress</th>
<th>Console Switch</th>
<th>I/O Stress</th>
<th>ProcFS Stress</th>
<th>Fork Stress</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monolithic</td>
<td>18512</td>
<td>6487</td>
<td>614</td>
<td>27596</td>
<td>3084</td>
<td>295</td>
<td></td>
</tr>
<tr>
<td>Low-Latency</td>
<td>63</td>
<td>6831</td>
<td>686</td>
<td>38</td>
<td>2904</td>
<td>332</td>
<td></td>
</tr>
<tr>
<td>Preemptable</td>
<td>17467</td>
<td>6012</td>
<td>213</td>
<td>187</td>
<td>31</td>
<td>329</td>
<td></td>
</tr>
<tr>
<td>Preemptable Lock-Breaking</td>
<td>54</td>
<td>6925</td>
<td>207</td>
<td>162</td>
<td>24</td>
<td>314</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. OS non-preemptable section latencies (in μs) for different kernels under different loads (test run for 25 seconds).

<table>
<thead>
<tr>
<th></th>
<th>Memory Stress</th>
<th>I/O Stress</th>
<th>ProcFS Stress</th>
<th>Fork Stress</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monolithic</td>
<td>18516</td>
<td>28314</td>
<td>3563</td>
<td>617</td>
</tr>
<tr>
<td>Low-Latency</td>
<td>293</td>
<td>292</td>
<td>3379</td>
<td>596</td>
</tr>
<tr>
<td>Preemptable</td>
<td>18848</td>
<td>392</td>
<td>224</td>
<td>645</td>
</tr>
<tr>
<td>Preemptable Lock-Breaking</td>
<td>239</td>
<td>322</td>
<td>231</td>
<td>537</td>
</tr>
</tbody>
</table>

Table 2. OS non-preemptable section latencies (in μs) for different kernels under different loads (tests run for 10 hours).
Figure 4. CDF of the latency measured on different versions of Linux (with high resolution timers). This test is performed with the I/O stress in background.

Figure 5. Audio/Video Skew on standard Linux. Heavy kernel load is run in the background.

Figure 6. Audio/Video Skew for lock-breaking preemptable Linux with high resolution timers. Heavy kernel load is run in the background. The Audio/Video skew is clustered around 0, and the maximum skew is less than 400µs (note that the scale is different from Figure 5).
Figure 7. Inter-Frame times for standard Linux. Heavy kernel load is run in the background.

Figure 8. Inter-Frame times for lock-breaking preemptable Linux with high resolution timers. Heavy kernel load is run in the background.