Real-Time Performance of Linux

- **Among others: "A Measurement-Based Analysis of the Real-Time Performance of Linux" (**L. Abeni , A. Goel, C. Krasic, J. Snow, J. Walpole) [RTAS 2002]

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^{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^{tick} = 10ms.
- How about decreasing T^{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. amound 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

```
void prune_dcache(int count)
{
        spin_lock(&dcache_lock);
        for (;;) {
            struct dentry *dentry;
            struct list_head *tmp;

            tmp = dentry_unused.prev;

            if (tmp == &dentry_unused)
                    break;
            list_del_init(tmp);
            dentry = list_entry(tmp, struct dentry, d_lru);

            /* If the dentry was recently referenced, don't free it. */
            if (dentry->d_vfs_flags & DCACHE_REFERENCED) {
                    dentry->d_vfs_flags &= ~DCACHE_REFERENCED;
                    list_add(&dentry->d_lru, &dentry_unused);
                    continue;
            }
            dentry_stat.nr_unused--;

            /* Unused dentry with a count? */
            if (atomic_read(&dentry->d_count))
                    BUG();

            prune_one_dentry(dentry);
            if (!--count)
                    break;
        }
        spin_unlock(&dcache_lock);
}
```

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

```
void prune_dcache(int count)
{
        DEFINE_RESCHED_COUNT;

redo:
        spin_lock(&dcache_lock);
        for (;;) {
                struct dentry *dentry;
                struct list_head *tmp;

                if (TEST_RESCHED_COUNT(100)) {
                    RESET_RESCHED_COUNT();
                    if (conditional_schedule_needed()) {
                            spin_unlock(&dcache_lock);
                            unconditional_schedule();
                            goto redo;
                    }
                }

                tmp = dentry_unused.prev;

                if (tmp == &dentry_unused)
                    break;
                list_del_init(tmp);
                dentry = list_entry(tmp, struct dentry, d_lru);

                /* If the dentry was recently referenced, don't free it. */
                if (dentry->d_vfs_flags & DCACHE_REFERENCED) {
                    dentry->d_vfs_flags &= ~DCACHE_REFERENCED;
                    list_add(&dentry->d_lru, &dentry_unused);
                    continue;
                }
                dentry_stat.nr_unused--;

                /* Unused dentry with a count? */
                if (atomic_read(&dentry->d_count))
                    BUG();

                prune_one_dentry(dentry);
                if (!--count)
                    break;
        }
        spin_unlock(&dcache_lock);
}
```

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()` ?)

Timer Latency

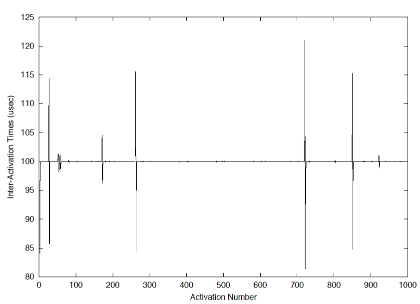


Figure 1. Inter-Activation times for a task that is woken up by a periodic signal with period $100\mu s$ on a high resolution timer Linux.

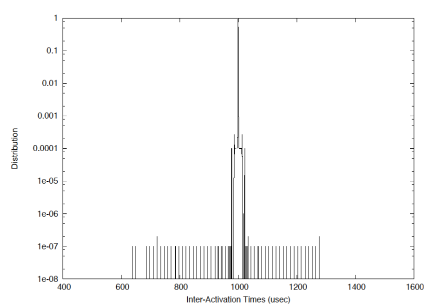


Figure 2. PDF of the difference between inter-activation times and period, when $T = 1000\mu s$.

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.
- **Procfs 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

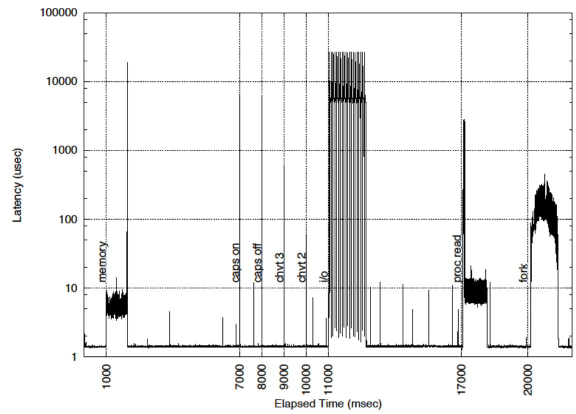
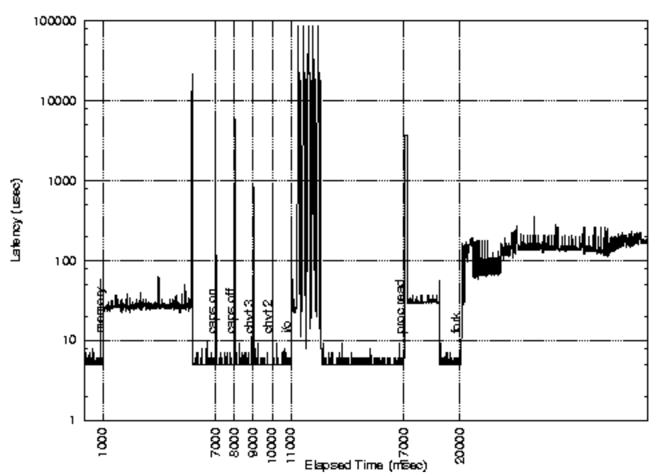


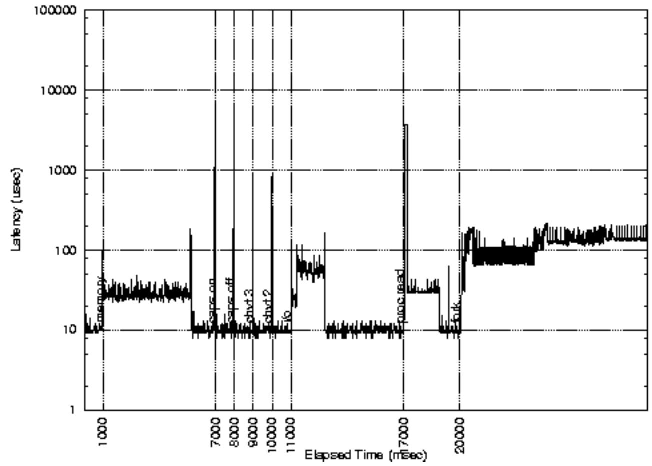
Figure 3. OS non-preemptable section latency measured on a high-resolution timer Linux. This test is performed with heavy background load.

Background Load Tests



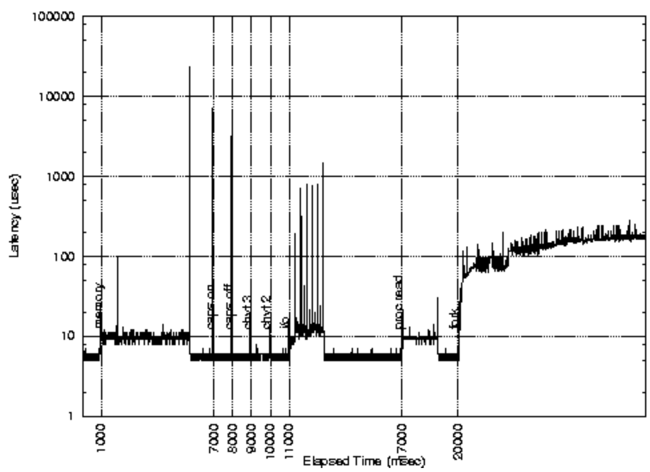
Standard Linux

Background Load Tests



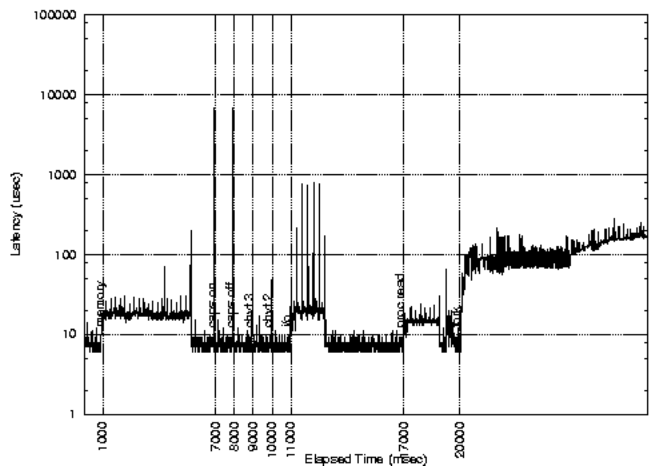
Low-Latency Kernel

Background Load Tests



Preemptable Kernel

Background Load Tests



Lock-Breaking Preemptable Kernel

© R. Bettati

OS Non-Preemptable Portion Latency

	Memory Stress	Caps-Lock Caps-Lock	Console Switch	I/O Stress	Procfs Stress	Fork Stress
Monolithic	18212	6487	614	27596	3084	295
Low-Latency	63	6831	686	38	2904	332
Preemptable	17467	6912	213	187	31	329
Preemptable Lock-Breaking	54	6525	207	162	24	314

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

	Memory Stress	I/O Stress	ProcFS Stress	Fork Stress
Monolithic	18956	28314	3563	617
Low-Latency	293	292	3379	596
Preemptable	18848	392	224	645
Preemptable Lock-Breaking	239	322	231	537

Table 2. OS non-preemptable section latencies (in μs) for different kernels under different loads (tests run for 10 hours).

© R. Bettati

Non-Preemptable Portion Latency

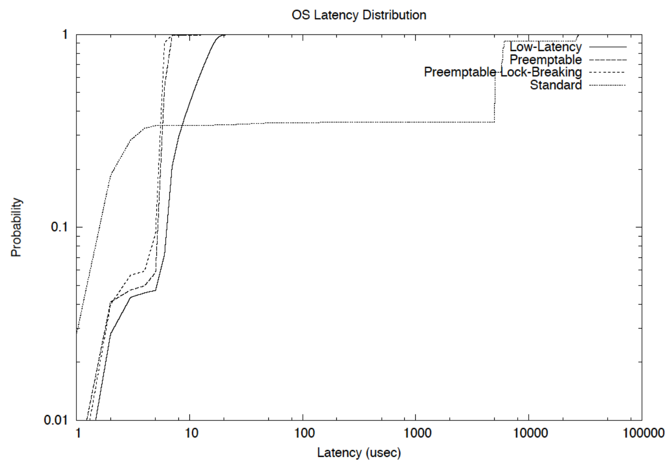


Figure 4. CDF of the latency measured on different versions of Linux (with high resolution timers). This test is performed with the I/O stress in background.

Latencies

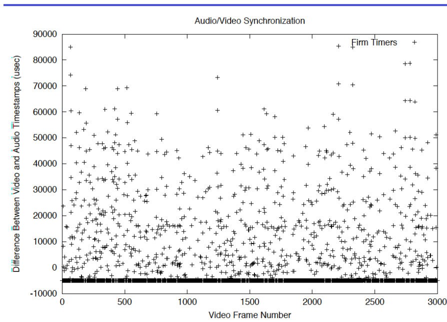


Figure 5. Audio/Video Skew on standard Linux. Heavy kernel load is run in the background.

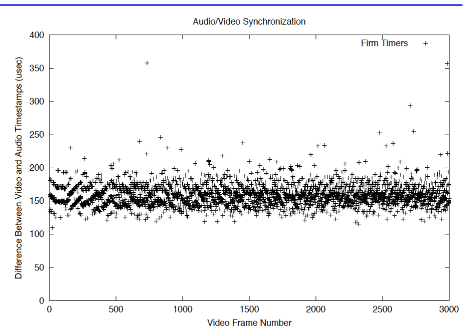


Figure 6. Audio/Video Skew for lock-breaking preemptable Linux with high resolution timers. Heavy kernel load is run in the background. The Audio/Video skew is clustered around 0**, and the maximum skew is less than** $400us$ **(note that the scale is different from Figure 5).**

Inter Frame Times

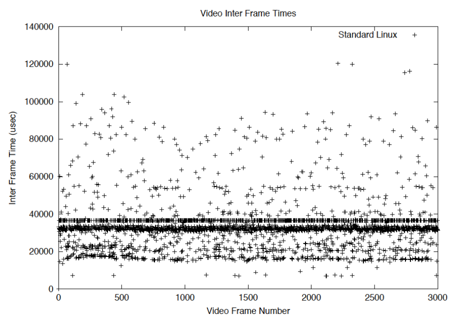


Figure 7. Inter-Frame times for standard Linux. Heavy kernel load is run in the background.

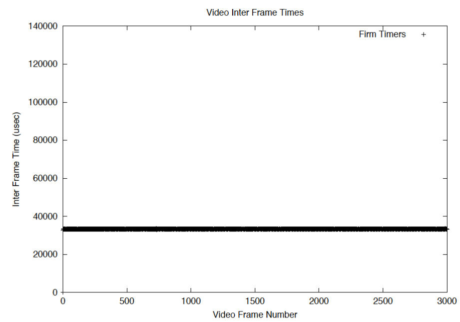


Figure 8. Inter-Frame times for lock-breaking preemptable Linux with high resolution timers. Heavy kernel load is run in the background.