CPU load

Linux exports various bits of information via `/proc/stat` and `/proc/uptime` that userland tools, such as top(1), use to calculate the average time system spent in a particular state, for example:

```
$ iostat
Linux 2.6.18.3-exp (linmac)     02/20/2007

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          10.01    0.00    2.92    5.44    0.00   81.63

...
```

Here the system thinks that over the default sampling period the system spent 10.01% of the time doing work in user space, 2.92% in the kernel, and was overall 81.63% of the time idle.

In most cases the `/proc/stat` information reflects the reality quite closely, however due to the nature of how/when the kernel collects this data sometimes it can not be trusted at all.

So how is this information collected? Whenever timer interrupt is signalled the kernel looks what kind of task was running at this moment and increments the counter that corresponds to this tasks kind/state. The problem with this is that the system could have switched between various states multiple times between two timer interrupts yet the counter is incremented only for the last state.

Example

If we imagine the system with one task that periodically burns cycles in the following manner:

```
 time line between two timer interrupts
|--------------------------------------|
 ^                                      ^
 |_ something begins working            |
                                        |_ something goes to sleep
                                       (only to be awaken quite soon)
```

In the above situation the system will be 0% loaded according to the `/proc/stat` (since the timer interrupt will always happen when the system is executing the idle handler), but in reality the load is closer to 99%.

One can imagine many more situations where this behavior of the kernel will lead to quite erratic information inside `/proc/stat`:

```
/* gcc -o hog smallhog.c */
#include <time.h>
#include <limits.h>
#include <signal.h>
#include <sys/time.h>
#define HIST 10

static volatile sig_atomic_t stop;

static void sighandler(int signr)
{
        (void) signr;
        stop = 1;
}

static unsigned long hog (unsigned long niters)
{
        stop = 0;
        while (!stop && --niters);
        return niters;
}

int main (void)
{
        int i;
        struct itimerval it = {
                .it_interval = { .tv_sec = 0, .tv_usec = 1 },
                .it_value    = { .tv_sec = 0, .tv_usec = 1 } };
        sigset_t set;
        unsigned long v[HIST];
        double tmp = 0.0;
        unsigned long n;
        signal(SIGALRM, &sighandler);
        setitimer(ITIMER_REAL, &it, NULL);

        hog (ULONG_MAX);
        for (i = 0; i < HIST; ++i) v[i] = ULONG_MAX - hog(ULONG_MAX);
        for (i = 0; i < HIST; ++i) tmp += v[i];
```

```
        tmp /= HIST;
        n = tmp - (tmp / 3.0);

        sigemptyset(&set);
        sigaddset(&set, SIGALRM);

        for (;;) {
                hog(n);
                sigwait(&set, &i);
        }
        return 0;
}
```

References

- https://lore.kernel.org/r/loom.20070212T063225-663@post.gmane.org
- Documentation/filesystems/proc.rst (1.8)

Thanks

Con Kolivas, Pavel Machek