

Bug: 2.6.6 Process accounting fails to account for small time slice loads

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2004-05/4313.html>

From: michael private (zbot666_at_yahoo.com)

Date: 05/21/04

Date: Fri, 21 May 2004 10:49:51 -0700 (PDT)

To: linux-kernel@vger.kernel.org

Situation:

For a program which runs a repetitive processing task followed by `nanosleep` or a `pthread_cond_timedwait`, if the processing task load is completed and the sleep state is entered in less than 1 jiffy (OS tick), there is no charge made to the process for CPU utilization under 2.6.6.

The CPU utilization, or lack thereof, can be observed with `times(2)`, `clock(3)`, or procs tools like `top`.

Under 2.4.20, normal CPU utilization is reported.

Both test boxes are dual processor XEON's running at 2.8 Ghz with hyperthreading enabled. The 2.4.20 box was built with `HZ` set to 1000, and both the low latency and pre-emption patch (assuming low latency and pre-emption was not the default -- will look this up further on request). The 2.6.6 box is running with the stock settings.

Below is a small test program which shows the differencing behavior for 2.4.20 and 2.6.6. At the bottom are run results.

The program runs a test load for approximately 0.5 milliseconds, followed by a call to `nanosleep` for 0.4 milliseconds. Note that the wakeup from `nanosleep` is automatically scheduled for two ticks in the future on 2.4.20, and for either 2 or three ticks into the future on 2.6.6, as has been discussed in other threads. Hence, each loop has a 1/2 tick load followed by a sleep period of 1.5 to 2.5 ticks. This combination of testload followed by `nanosleep` is run

Linux-Kernel: Bug: 2.6.6 Process accounting fails to account for small time slice loads

1000 times. The printout shows the results from times() and clock() -- nothing charged after 1000 loops for 2.6.6, while 0.5 seconds charged after 1000 loops for 2.4.20.

To calibrate the runload loop, adjust the constant millisec smaller for machines with a slower clock cycle, i.e. 100000 works for a 2.8 Ghz xeon, probably 25000 will work for a 700 Mhz PIII machine. Set adjust this until the "Benchmark" runs in 1/2 second, and then you should get similar results.

Thank you collectively.

-Mike

```
//compile with gcc -lrt test.c
```

```
#include <stdio.h>
#include <string.h>
#include <sys/times.h>
#include <unistd.h>
#include <time.h>

/*****

void runload();
double mSec = 1e6;
double millisec = 100000; //this constant
appropriate on 2.8 Ghz Xeon

/*****

inline double doubleTime()
{
    double t;
    struct timespec ts;

    clock_gettime(CLOCK_REALTIME, &ts);
    t = (double) ts.tv_sec + (1e-9) * (double)
ts.tv_nsec;
    return(t);
}
/*****

typedef struct tStruct
{
    double t;
    clock_t clockt;
    struct tms tms;
} TSTRUCT;
```

Linux-Kernel: Bug: 2.6.6 Process accounting fails to account for small time slice loads

```
void setT(TSTRUCT *ts)
{
    times(&ts->tms);
    ts->clockt = clock();
    ts->t = doubleTime();
}

//*****

void printDt(char * msg, int id, TSTRUCT *t1, TSTRUCT
*t2)
{
    const double k = 1.0 / CLOCKS_PER_SEC;
    const double k2 = 0.01;

    double dclock_t1, dclock_t2;

    printf("%s(%d) t1=%f t2=%f dt=%f\n", msg, id,
t1->t,t2->t,t2->t-t1->t);
    dclock_t1 = t1->clockt * k;
    dclock_t2 = t2->clockt * k;
    printf("%s(%d) dct1=%f dct2=%f dcdt=%f\n", msg,
id, dclock_t1,dclock_t2,dclock_t2-dclock_t1);
    printf("%s(%d) ut1=%f ut2=%f dut=%f\n", msg, id,
t1->tms.tms_utime*k2, t2->tms.tms_utime*k2,
    k2*(t2->tms.tms_utime - t1->tms.tms_utime));
    printf("%s(%d) st1=%f st2=%f dst=%f\n", msg, id,
t1->tms.tms_stime*k2, t2->tms.tms_stime*k2,
    k2*(t2->tms.tms_stime - t1->tms.tms_stime));
}

//*****

main()
{
    TSTRUCT t1, t2;
    struct timespec nSleep;
    int i,j = 0;

    setT(&t1);
    runload((int) (500.0 * millisec));
    setT(&t2);
    printDt("BenchMark", 0, &t1, &t2);

    nSleep.tv_sec = 0;
    nSleep.tv_nsec = 0.4 * mSec;

    while(1)
    {
        setT(&t1);
        for (i=0; i<1000; ++i)
```

Linux–Kernel: Bug: 2.6.6 Process accounting fails to account for small time slice loads

```
{
runload((int) (0.5 * millisec));

    nanosleep(&nSleep, NULL);
}
setT(&t2);
printDt("1000 .5 mS runs", j, &t1, &t2);
j = j + 1;
}

}

//*****

void runload(int count)
{
    int i;
    double x = -14400;

    for (i=0; i<count; ++i)
        {
            x += 2*i;
        }
}

//---END OF PROGRAM---
```

On 2.6.6 Box

```
BenchMark(0) t1=1085156944.775533 t2=1085156945.283144
dt=0.507611
BenchMark(0) dct1=0.000000 dct2=0.500000 dcdt=0.500000
BenchMark(0) ut1=0.000000 ut2=0.500000 dut=0.500000
BenchMark(0) st1=0.000000 st2=0.000000 dst=0.000000
1000 .5 mS runs(0) t1=1085156945.283247
t2=1085156947.885763 dt=2.602516
1000 .5 mS runs(0) dct1=0.500000 dct2=0.510000
dcdt=0.010000
1000 .5 mS runs(0) ut1=0.500000 ut2=0.510000
dut=0.010000
1000 .5 mS runs(0) st1=0.000000 st2=0.000000
dst=0.000000
1000 .5 mS runs(1) t1=1085156947.885794
t2=1085156950.885107 dt=2.999313
1000 .5 mS runs(1) dct1=0.510000 dct2=0.510000
dcdt=0.000000
1000 .5 mS runs(1) ut1=0.510000 ut2=0.510000
dut=0.000000
1000 .5 mS runs(1) st1=0.000000 st2=0.000000
```

Bug: 2.6.6 Process accounting fails to account for small time slice loads

Linux–Kernel: Bug: 2.6.6 Process accounting fails to account for small time slice loads

```
dst=0.000000
1000 .5 mS runs(2) t1=1085156950.885139
t2=1085156953.347563 dt=2.462424
1000 .5 mS runs(2) dct1=0.510000 dct2=0.510000
dcdt=0.000000
1000 .5 mS runs(2) ut1=0.510000 ut2=0.510000
dut=0.000000
1000 .5 mS runs(2) st1=0.000000 st2=0.000000
dst=0.000000
1000 .5 mS runs(3) t1=1085156953.347586
t2=1085156955.347127 dt=1.999541
1000 .5 mS runs(3) dct1=0.510000 dct2=0.510000
dcdt=0.000000
1000 .5 mS runs(3) ut1=0.510000 ut2=0.510000
dut=0.000000
1000 .5 mS runs(3) st1=0.000000 st2=0.000000
dst=0.000000
```

On 2.4.20 Box

```
BenchMark(0) t1=1085156916.233624 t2=1085156916.704806
dt=0.471182
BenchMark(0) dct1=0.000000 dct2=0.450000 dcdt=0.450000
BenchMark(0) ut1=0.000000 ut2=0.450000 dut=0.450000
BenchMark(0) st1=0.000000 st2=0.000000 dst=0.000000
1000 .5 mS runs(0) t1=1085156916.704942
t2=1085156918.718239 dt=2.013297
1000 .5 mS runs(0) dct1=0.450000 dct2=1.090000
dcdt=0.640000
1000 .5 mS runs(0) ut1=0.450000 ut2=1.090000
dut=0.640000
1000 .5 mS runs(0) st1=0.000000 st2=0.000000
dst=0.000000
1000 .5 mS runs(1) t1=1085156918.718296
t2=1085156920.721021 dt=2.002725
1000 .5 mS runs(1) dct1=1.090000 dct2=1.490000
dcdt=0.400000
1000 .5 mS runs(1) ut1=1.090000 ut2=1.490000
dut=0.400000
1000 .5 mS runs(1) st1=0.000000 st2=0.000000
dst=0.000000
1000 .5 mS runs(2) t1=1085156920.721129
t2=1085156922.725788 dt=2.004659
1000 .5 mS runs(2) dct1=1.490000 dct2=1.810000
dcdt=0.320000
1000 .5 mS runs(2) ut1=1.490000 ut2=1.810000
dut=0.320000
1000 .5 mS runs(2) st1=0.000000 st2=0.000000
dst=0.000000
```

Do you Yahoo!?

Yahoo! Domains – Claim yours for only \$14.70/year

<http://smallbusiness.promotions.yahoo.com/offer>

–

To unsubscribe from this list: send the line "unsubscribe linux–kernel" in the body of a message to majordomo@vger.kernel.org

More majordomo info at <http://vger.kernel.org/majordomo–info.html>

Please read the FAQ at <http://www.tux.org/lkml/>