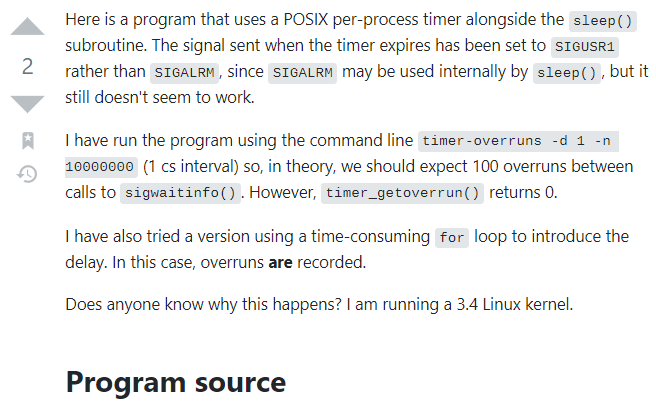
REF:

<https://stackoverflow.com/questions/11379925/timer-getoverrun-doesnt-behave-as-expected-when-using-sleep>





/\*

\* timer-overruns.c

\*/

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <signal.h>

#include <time.h>

// Signal to be used for timer expirations

#define TIMER\_SIGNAL SIGUSR1

int main(int argc, char \*\*argv) {

int opt;

int d = 0;

int r = 0; // Repeat indefinitely

struct itimerspec its;

its.it\_interval.tv\_sec = 0;

its.it\_interval.tv\_nsec = 0;

// Parse arguments

while ((opt = getopt(argc, argv, "d:r:s:n:")) != -1) {

switch (opt) {

case 'd': // Delay before calling sigwaitinfo()

d = atoi(optarg);

break;

case 'r': // Number of times to call sigwaitinfo()

r = atoi(optarg);

break;

case 's': // Timer interval (seconds)

its.it\_interval.tv\_sec = its.it\_value.tv\_sec = atoi(optarg);

break;

case 'n': // Timer interval (nanoseconds)

its.it\_interval.tv\_nsec = its.it\_value.tv\_nsec = atoi(optarg);

break;

default: /\* '?' \*/

fprintf(stderr,

"Usage: %s [-d signal\_accept\_delay] [-r repetitions] [-s interval\_seconds] [-n interval\_nanoseconds]\n",

argv[0]);

exit(EXIT\_FAILURE);

}

}

// Check sanity of command line arguments

short e = 0;

if (d < 0) {

fprintf(stderr, "Delay (-d) cannot be negative!\n");

e++;

}

if (r < 0) {

fprintf(stderr, "Number of repetitions (-r) cannot be negative!\n");

e++;

}

if (its.it\_interval.tv\_sec < 0) {

fprintf(stderr, "Interval seconds value (-s) cannot be negative!\n");

e++;

}

if (its.it\_interval.tv\_nsec < 0) {

fprintf(stderr, "Interval nanoseconds value (-n) cannot be negative!\n");

e++;

}

if (its.it\_interval.tv\_nsec > 999999999) {

fprintf(stderr, "Interval nanoseconds value (-n) must be < 1 second.\n");

e++;

}

if (e > 0)

exit(EXIT\_FAILURE);

// Set default values if not specified

if (its.it\_interval.tv\_sec == 0 && its.it\_interval.tv\_nsec == 0) {

its.it\_interval.tv\_sec = its.it\_value.tv\_sec = 1;

its.it\_value.tv\_nsec = 0;

}

printf("Running with timer delay %d.%09d seconds\n",

(int) its.it\_interval.tv\_sec, (int) its.it\_interval.tv\_nsec);

// Will be waiting for signals synchronously, so block the one in use.

sigset\_t sigset;

sigemptyset(&sigset);

sigaddset(&sigset, TIMER\_SIGNAL);

sigprocmask(SIG\_BLOCK, &sigset, NULL );

// Create and arm the timer

struct sigevent sev;

timer\_t timer;

sev.sigev\_notify = SIGEV\_SIGNAL;

sev.sigev\_signo = TIMER\_SIGNAL;

sev.sigev\_value.sival\_ptr = timer;

timer\_create(CLOCK\_REALTIME, &sev, &timer);

timer\_settime(timer, TIMER\_ABSTIME, &its, NULL );

// Signal handling loop

int overruns;

siginfo\_t si;

// Make the loop infinite if r = 0

if (r == 0)

r = -1;

while (r != 0) {

// Sleeping should cause overruns

if (d > 0)

sleep(d);

sigwaitinfo(&sigset, &si);

// Check that the signal is from the timer

if (si.si\_code != SI\_TIMER)

continue;

overruns = timer\_getoverrun(timer);

if (overruns > 0) {

printf("Timer overrun occurred for %d expirations.\n", overruns);

}

// Decrement r if not repeating indefinitely

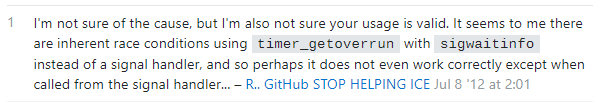
if (r > 0)

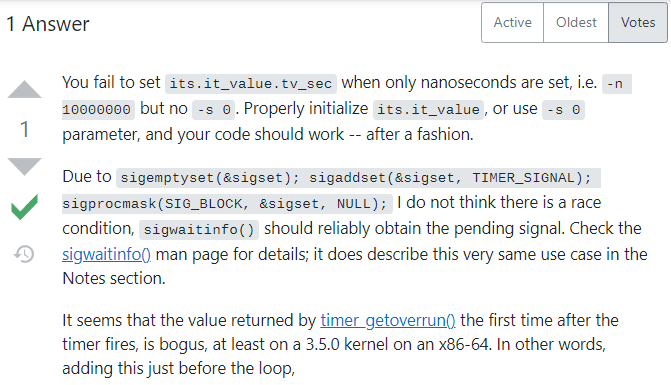
r--;

}

return EXIT\_SUCCESS;

}





printf("Overruns: %d\n",

timer\_getoverrun(timer));

sigwaitinfo(&sigset, &si);

printf("Overruns: %d (%d)\n",

timer\_getoverrun(timer), si.si\_overrun);

sigwaitinfo(&sigset, &si);

printf("Overruns: %d (%d)\n",

timer\_getoverrun(timer), si.si\_overrun);

