```
/*
 * Tests for prctl(PR_GET_TSC, ...) / prctl(PR_SET_TSC, ...)
 *
 * Tests if the control register is updated correctly
 * at context switches
 *
 * Warning: this test will cause a very high load for a few seconds
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <inttypes.h>
#include <wait.h>


#include <sys/prctl.h>
#include <linux/prctl.h>

/* Get/set the process' ability to use the timestamp counter instruction */
#ifndef PR_GET_TSC
#define PR_GET_TSC 25
#define PR_SET_TSC 26
# define PR_TSC_ENABLE      1   /* allow the use of the timestamp counter */
# define PR_TSC_SIGSEGV     2   /* throw a SIGSEGV instead of reading the TSC */
#endif

uint64_t rdtsc() {
uint32_t lo, hi;
/* We cannot use "=A", since this would use %rax on x86_64 */
__asm__ __volatile__ ("rdtsc" : "=a" (lo), "=d" (hi));
return (uint64_t)hi << 32 | lo;
}

void sigsegv_expect(int sig)
{
    /* */
}

void segvtask(void)
{
    if (prctl(PR_SET_TSC, PR_TSC_SIGSEGV) < 0)
    {
        perror("prctl");
        exit(0);
    }
    signal(SIGSEGV, sigsegv_expect);
    alarm(10);
    rdtsc();
    fprintf(stderr, "FATAL ERROR, rdtsc() succeeded while disabled\n");
    exit(0);
}


void sigsegv_fail(int sig)
{
    fprintf(stderr, "FATAL ERROR, rdtsc() failed while enabled\n");
    exit(0);
}

void rdtsctask(void)
{
    if (prctl(PR_SET_TSC, PR_TSC_ENABLE) < 0)
    {
```

```
        perror("prctl");
        exit(0);
    }
    signal(SIGSEGV, sigsegv_fail);
    alarm(10);
    for(;;) rdtsc();
}


int main(int argc, char **argv)
{
    int n_tasks = 100, i;

    fprintf(stderr, "[No further output means we're allright]\n");

    for (i=0; i<n_tasks; i++)
        if (fork() == 0)
        {
            if (i & 1)
                segvtask();
            else
                rdtsctask();
        }

    for (i=0; i<n_tasks; i++)
        wait(NULL);

    exit(0);
}
```