```
/*
 * dslm.c
 * Simple Disk Sleep Monitor
 *  by Bartek Kania
 * Licenced under the GPL
 */
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <time.h>
#include <string.h>
#include <signal.h>
#include <sys/ioctl.h>
#include <linux/hdreg.h>

#ifdef DEBUG
#define D(x) x
#else
#define D(x)
#endif

int endit = 0;

/* Check if the disk is in powersave-mode
 * Most of the code is stolen from hdparm.
 * 1 = active, 0 = standby/sleep, -1 = unknown */
static int check_powermode(int fd)
{
    unsigned char args[4] = {WIN_CHECKPOWERMODE1,0,0,0};
    int state;

    if (ioctl(fd, HDIO_DRIVE_CMD, &args)
    && (args[0] = WIN_CHECKPOWERMODE2) /* try again with 0x98 */
    && ioctl(fd, HDIO_DRIVE_CMD, &args)) {
    if (errno != EIO || args[0] != 0 || args[1] != 0) {
        state = -1; /* "unknown"; */
    } else
        state = 0; /* "sleeping"; */
    } else {
    state = (args[2] == 255) ? 1 : 0;
    }
    D(printf(" drive state is:  %d\n", state));

    return state;
}

static char *state_name(int i)
{
    if (i == -1) return "unknown";
    if (i == 0) return "sleeping";
    if (i == 1) return "active";

    return "internal error";
}

static char *myctime(time_t time)
{
    char *ts = ctime(&time);
    ts[strlen(ts) - 1] = 0;

    return ts;
}

static void measure(int fd)
```

```
{
    time_t start_time;
    int last_state;
    time_t last_time;
    int curr_state;
    time_t curr_time = 0;
    time_t time_diff;
    time_t active_time = 0;
    time_t sleep_time = 0;
    time_t unknown_time = 0;
    time_t total_time = 0;
    int changes = 0;
    float tmp;

    printf("Starting measurements\n");

    last_state = check_powermode(fd);
    start_time = last_time = time(0);
    printf("  System is in state %s\n\n", state_name(last_state));

    while(!endit) {
    sleep(1);
    curr_state = check_powermode(fd);

    if (curr_state != last_state || endit) {
        changes++;
        curr_time = time(0);
        time_diff = curr_time - last_time;

        if (last_state == 1) active_time += time_diff;
        else if (last_state == 0) sleep_time += time_diff;
        else unknown_time += time_diff;

        last_state = curr_state;
        last_time = curr_time;

        printf("%s: State-change to %s\n", myctime(curr_time),
           state_name(curr_state));
    }
    }
    changes--; /* Compensate for SIGINT */

    total_time = time(0) - start_time;
    printf("\nTotal running time:  %lus\n", curr_time - start_time);
    printf(" State changed %d times\n", changes);

    tmp = (float)sleep_time / (float)total_time * 100;
    printf(" Time in sleep state:   %lus (%.2f%%)\n", sleep_time, tmp);
    tmp = (float)active_time / (float)total_time * 100;
    printf(" Time in active state:  %lus (%.2f%%)\n", active_time, tmp);
    tmp = (float)unknown_time / (float)total_time * 100;
    printf(" Time in unknown state: %lus (%.2f%%)\n", unknown_time, tmp);
}

static void ender(int s)
{
    endit = 1;
}

static void usage(void)
{
    puts("usage: dslm [-w <time>] <disk>");
    exit(0);
}

int main(int argc, char **argv)
```

```
{
    int fd;
    char *disk = 0;
    int settle_time = 60;

    /* Parse the simple command-line */
    if (argc == 2)
    disk = argv[1];
    else if (argc == 4) {
    settle_time = atoi(argv[2]);
    disk = argv[3];
    } else
    usage();

    if (!(fd = open(disk, O_RDONLY|O_NONBLOCK))) {
    printf("Can't open %s, because: %s\n", disk, strerror(errno));
    exit(-1);
    }

    if (settle_time) {
    printf("Waiting %d seconds for the system to settle down to "
           "'normal'\n", settle_time);
    sleep(settle_time);
    } else
    puts("Not waiting for system to settle down");

    signal(SIGINT, ender);

    measure(fd);

    close(fd);

    return 0;
}
```