```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <memory.h>
#include <malloc.h>
#include <time.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>
#include <errno.h>
#include <sys/time.h>
#include <linux/hpet.h>


extern void hpet_open_close(int, const char **);
extern void hpet_info(int, const char **);
extern void hpet_poll(int, const char **);
extern void hpet_fasync(int, const char **);
extern void hpet_read(int, const char **);

#include <sys/poll.h>
#include <sys/ioctl.h>

struct hpet_command {
    char        *command;
    void        (*func)(int argc, const char ** argv);
} hpet_command[] = {
    {
        "open-close",
        hpet_open_close
    },
    {
        "info",
        hpet_info
    },
    {
        "poll",
        hpet_poll
    },
    {
        "fasync",
        hpet_fasync
    },
};

int
main(int argc, const char ** argv)
{
    int i;

    argc--;
    argv++;

    if (!argc) {
        fprintf(stderr, "-hpet: requires command\n");
        return -1;
    }


    for (i = 0; i < (sizeof (hpet_command) / sizeof (hpet_command[0])); i++)
        if (!strcmp(argv[0], hpet_command[i].command)) {
            argc--;
            argv++;
```

```
            fprintf(stderr, "-hpet: executing %s\n",
                hpet_command[i].command);
            hpet_command[i].func(argc, argv);
            return 0;
        }

    fprintf(stderr, "do_hpet: command %s not implemented\n", argv[0]);

    return -1;
}

void
hpet_open_close(int argc, const char **argv)
{
    int fd;

    if (argc != 1) {
        fprintf(stderr, "hpet_open_close: device-name\n");
        return;
    }

    fd = open(argv[0], O_RDONLY);
    if (fd < 0)
        fprintf(stderr, "hpet_open_close: open failed\n");
    else
        close(fd);

    return;
}

void
hpet_info(int argc, const char **argv)
{
}

void
hpet_poll(int argc, const char **argv)
{
    unsigned long       freq;
    int         iterations, i, fd;
    struct pollfd       pfd;
    struct hpet_info    info;
    struct timeval      stv, etv;
    struct timezone     tz;
    long            usec;

    if (argc != 3) {
        fprintf(stderr, "hpet_poll: device-name freq iterations\n");
        return;
    }

    freq = atoi(argv[1]);
    iterations = atoi(argv[2]);

    fd = open(argv[0], O_RDONLY);

    if (fd < 0) {
        fprintf(stderr, "hpet_poll: open of %s failed\n", argv[0]);
        return;
    }

    if (ioctl(fd, HPET_IRQFREQ, freq) < 0) {
        fprintf(stderr, "hpet_poll: HPET_IRQFREQ failed\n");
        goto out;
    }
```

```
    if (ioctl(fd, HPET_INFO, &info) < 0) {
        fprintf(stderr, "hpet_poll: failed to get info\n");
        goto out;
    }

    fprintf(stderr, "hpet_poll: info.hi_flags 0x%lx\n", info.hi_flags);

    if (info.hi_flags && (ioctl(fd, HPET_EPI, 0) < 0)) {
        fprintf(stderr, "hpet_poll: HPET_EPI failed\n");
        goto out;
    }

    if (ioctl(fd, HPET_IE_ON, 0) < 0) {
        fprintf(stderr, "hpet_poll, HPET_IE_ON failed\n");
        goto out;
    }

    pfd.fd = fd;
    pfd.events = POLLIN;

    for (i = 0; i < iterations; i++) {
        pfd.revents = 0;
        gettimeofday(&stv, &tz);
        if (poll(&pfd, 1, -1) < 0)
            fprintf(stderr, "hpet_poll: poll failed\n");
        else {
            long    data;

            gettimeofday(&etv, &tz);
            usec = stv.tv_sec * 1000000 + stv.tv_usec;
            usec = (etv.tv_sec * 1000000 + etv.tv_usec) - usec;

            fprintf(stderr,
                "hpet_poll: expired time = 0x%lx\n", usec);

            fprintf(stderr, "hpet_poll: revents = 0x%x\n",
                pfd.revents);

            if (read(fd, &data, sizeof(data)) != sizeof(data)) {
                fprintf(stderr, "hpet_poll: read failed\n");
            }
            else
                fprintf(stderr, "hpet_poll: data 0x%lx\n",
                    data);
        }
    }

out:
    close(fd);
    return;
}

static int hpet_sigio_count;

static void
hpet_sigio(int val)
{
    fprintf(stderr, "hpet_sigio: called\n");
    hpet_sigio_count++;
}

void
hpet_fasync(int argc, const char **argv)
{
    unsigned long       freq;
    int         iterations, i, fd, value;
```

```
    sig_t           oldsig;
    struct hpet_info    info;

    hpet_sigio_count = 0;
    fd = -1;

    if ((oldsig = signal(SIGIO, hpet_sigio)) == SIG_ERR) {
        fprintf(stderr, "hpet_fasync: failed to set signal handler\n");
        return;
    }

    if (argc != 3) {
        fprintf(stderr, "hpet_fasync: device-name freq iterations\n");
        goto out;
    }

    fd = open(argv[0], O_RDONLY);

    if (fd < 0) {
        fprintf(stderr, "hpet_fasync: failed to open %s\n", argv[0]);
        return;
    }


    if ((fcntl(fd, F_SETOWN, getpid()) == 1) ||
        ((value = fcntl(fd, F_GETFL)) == 1) ||
        (fcntl(fd, F_SETFL, value | O_ASYNC) == 1)) {
        fprintf(stderr, "hpet_fasync: fcntl failed\n");
        goto out;
    }

    freq = atoi(argv[1]);
    iterations = atoi(argv[2]);

    if (ioctl(fd, HPET_IRQFREQ, freq) < 0) {
        fprintf(stderr, "hpet_fasync: HPET_IRQFREQ failed\n");
        goto out;
    }

    if (ioctl(fd, HPET_INFO, &info) < 0) {
        fprintf(stderr, "hpet_fasync: failed to get info\n");
        goto out;
    }

    fprintf(stderr, "hpet_fasync: info.hi_flags 0x%lx\n", info.hi_flags);

    if (info.hi_flags && (ioctl(fd, HPET_EPI, 0) < 0)) {
        fprintf(stderr, "hpet_fasync: HPET_EPI failed\n");
        goto out;
    }

    if (ioctl(fd, HPET_IE_ON, 0) < 0) {
        fprintf(stderr, "hpet_fasync, HPET_IE_ON failed\n");
        goto out;
    }

    for (i = 0; i < iterations; i++) {
        (void) pause();
        fprintf(stderr, "hpet_fasync: count = %d\n", hpet_sigio_count);
    }

out:
    signal(SIGIO, oldsig);

    if (fd >= 0)
        close(fd);
```

```
    return;
}
```