```
/*
 * This program demonstrates how the various time stamping features in
 * the Linux kernel work. It emulates the behavior of a PTP
 * implementation in stand-alone master mode by sending PTPv1 Sync
 * multicasts once every second. It looks for similar packets, but
 * beyond that doesn't actually implement PTP.
 *
 * Outgoing packets are time stamped with SO_TIMESTAMPING with or
 * without hardware support.
 *
 * Incoming packets are time stamped with SO_TIMESTAMPING with or
 * without hardware support, SIOCGSTAMP[NS] (per-socket time stamp) and
 * SO_TIMESTAMP[NS].
 *
 * Copyright (C) 2009 Intel Corporation.
 * Author: Patrick Ohly <patrick.ohly@intel.com>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms and conditions of the GNU General Public License,
 * version 2, as published by the Free Software Foundation.
 *
 * This program is distributed in the hope it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE. * See the GNU General Public License for
 * more details.
 *
 * You should have received a copy of the GNU General Public License along with
 * this program; if not, write to the Free Software Foundation, Inc.,
 * 51 Franklin St - Fifth Floor, Boston, MA 02110-1301 USA.
 */

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>

#include <sys/time.h>
#include <sys/socket.h>
#include <sys/select.h>
#include <sys/ioctl.h>
#include <arpa/inet.h>
#include <net/if.h>

#include <asm/types.h>
#include <linux/net_tstamp.h>
#include <linux/errqueue.h>

#ifndef SO_TIMESTAMPING
# define SO_TIMESTAMPING         37
# define SCM_TIMESTAMPING        SO_TIMESTAMPING
#endif

#ifndef SO_TIMESTAMPNS
# define SO_TIMESTAMPNS 35
#endif

#ifndef SIOCGSTAMPNS
# define SIOCGSTAMPNS 0x8907
#endif

#ifndef SIOCSHWTSTAMP
# define SIOCSHWTSTAMP 0x89b0
#endif

static void usage(const char *error)
{
```

```
    if (error)
        printf("invalid option: %s\n", error);
    printf("timestamping interface option*\n\n"
           "Options:\n"
           "  IP_MULTICAST_LOOP - looping outgoing multicasts\n"
           "  SO_TIMESTAMP - normal software time stamping, ms resolution\n"
           "  SO_TIMESTAMPNS - more accurate software time stamping\n"
           "  SOF_TIMESTAMPING_TX_HARDWARE - hardware time stamping of outgoing packets\n"
           "  SOF_TIMESTAMPING_TX_SOFTWARE - software fallback for outgoing packets\n"
           "  SOF_TIMESTAMPING_RX_HARDWARE - hardware time stamping of incoming packets\n"
           "  SOF_TIMESTAMPING_RX_SOFTWARE - software fallback for incoming packets\n"
           "  SOF_TIMESTAMPING_SOFTWARE - request reporting of software time stamps\n"
           "  SOF_TIMESTAMPING_SYS_HARDWARE - request reporting of transformed HW time stamps\n"
           "  SOF_TIMESTAMPING_RAW_HARDWARE - request reporting of raw HW time stamps\n"
           "  SIOCGSTAMP - check last socket time stamp\n"
           "  SIOCGSTAMPNS - more accurate socket time stamp\n");
    exit(1);
}

static void bail(const char *error)
{
    printf("%s: %s\n", error, strerror(errno));
    exit(1);
}

static const unsigned char sync[] = {
    0x00, 0x01, 0x00, 0x01,
    0x5f, 0x44, 0x46, 0x4c,
    0x54, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00,
    0x01, 0x01,

    /* fake uuid */
    0x00, 0x01,
    0x02, 0x03, 0x04, 0x05,

    0x00, 0x01, 0x00, 0x37,
    0x00, 0x00, 0x00, 0x08,
    0x00, 0x00, 0x00, 0x00,
    0x49, 0x05, 0xcd, 0x01,
    0x29, 0xb1, 0x8d, 0xb0,
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x01,

    /* fake uuid */
    0x00, 0x01,
    0x02, 0x03, 0x04, 0x05,

    0x00, 0x00, 0x00, 0x37,
    0x00, 0x00, 0x00, 0x04,
    0x44, 0x46, 0x4c, 0x54,
    0x00, 0x00, 0xf0, 0x60,
    0x00, 0x01, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x01,
    0x00, 0x00, 0xf0, 0x60,
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x04,
    0x44, 0x46, 0x4c, 0x54,
    0x00, 0x01,

    /* fake uuid */
    0x00, 0x01,
    0x02, 0x03, 0x04, 0x05,

    0x00, 0x00, 0x00, 0x00,
```

```
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00
};

static void sendpacket(int sock, struct sockaddr *addr, socklen_t addr_len)
{
    struct timeval now;
    int res;

    res = sendto(sock, sync, sizeof(sync), 0,
        addr, addr_len);
    gettimeofday(&now, 0);
    if (res < 0)
        printf("%s: %s\n", "send", strerror(errno));
    else
        printf("%ld.%06ld: sent %d bytes\n",
               (long)now.tv_sec, (long)now.tv_usec,
               res);
}

static void printpacket(struct msghdr *msg, int res,
            char *data,
            int sock, int recvmsg_flags,
            int siocgstamp, int siocgstampns)
{
    struct sockaddr_in *from_addr = (struct sockaddr_in *)msg->msg_name;
    struct cmsghdr *cmsg;
    struct timeval tv;
    struct timespec ts;
    struct timeval now;

    gettimeofday(&now, 0);

    printf("%ld.%06ld: received %s data, %d bytes from %s, %zu bytes control messages\n",
           (long)now.tv_sec, (long)now.tv_usec,
           (recvmsg_flags & MSG_ERRQUEUE) ? "error" : "regular",
           res,
           inet_ntoa(from_addr->sin_addr),
           msg->msg_controllen);
    for (cmsg = CMSG_FIRSTHDR(msg);
         cmsg;
         cmsg = CMSG_NXTHDR(msg, cmsg)) {
        printf("   cmsg len %zu: ", cmsg->cmsg_len);
        switch (cmsg->cmsg_level) {
        case SOL_SOCKET:
            printf("SOL_SOCKET ");
            switch (cmsg->cmsg_type) {
            case SO_TIMESTAMP: {
                struct timeval *stamp =
                    (struct timeval *)CMSG_DATA(cmsg);
                printf("SO_TIMESTAMP %ld.%06ld",
                       (long)stamp->tv_sec,
                       (long)stamp->tv_usec);
                break;
            }
            case SO_TIMESTAMPNS: {
                struct timespec *stamp =
                    (struct timespec *)CMSG_DATA(cmsg);
                printf("SO_TIMESTAMPNS %ld.%09ld",
                       (long)stamp->tv_sec,
                       (long)stamp->tv_nsec);
                break;
            }
            case SO_TIMESTAMPING: {
                struct timespec *stamp =
```

```
                    (struct timespec *)CMSG_DATA(cmsg);
                printf("SO_TIMESTAMPING ");
                printf("SW %ld.%09ld ",
                       (long)stamp->tv_sec,
                       (long)stamp->tv_nsec);
                stamp++;
                printf("HW transformed %ld.%09ld ",
                       (long)stamp->tv_sec,
                       (long)stamp->tv_nsec);
                stamp++;
                printf("HW raw %ld.%09ld",
                       (long)stamp->tv_sec,
                       (long)stamp->tv_nsec);
                break;
            }
            default:
                printf("type %d", cmsg->cmsg_type);
                break;
            }
            break;
        case IPPROTO_IP:
            printf("IPPROTO_IP ");
            switch (cmsg->cmsg_type) {
            case IP_RECVERR: {
                struct sock_extended_err *err =
                    (struct sock_extended_err *)CMSG_DATA(cmsg);
                printf("IP_RECVERR ee_errno '%s' ee_origin %d => %s",
                    strerror(err->ee_errno),
                    err->ee_origin,
#ifdef SO_EE_ORIGIN_TIMESTAMPING
                    err->ee_origin == SO_EE_ORIGIN_TIMESTAMPING ?
                    "bounced packet" : "unexpected origin"
#else
                    "probably SO_EE_ORIGIN_TIMESTAMPING"
#endif
                    );
                if (res < sizeof(sync))
                    printf(" => truncated data?!");
                else if (!memcmp(sync, data + res - sizeof(sync),
                            sizeof(sync)))
                    printf(" => GOT OUR DATA BACK (HURRAY!)");
                break;
            }
            case IP_PKTINFO: {
                struct in_pktinfo *pktinfo =
                    (struct in_pktinfo *)CMSG_DATA(cmsg);
                printf("IP_PKTINFO interface index %u",
                    pktinfo->ipi_ifindex);
                break;
            }
            default:
                printf("type %d", cmsg->cmsg_type);
                break;
            }
            break;
        default:
            printf("level %d type %d",
                cmsg->cmsg_level,
                cmsg->cmsg_type);
            break;
        }
        printf("\n");
    }

    if (siocgstamp) {
        if (ioctl(sock, SIOCGSTAMP, &tv))
```

```
            printf("   %s: %s\n", "SIOCGSTAMP", strerror(errno));
        else
            printf("SIOCGSTAMP %ld.%06ld\n",
                   (long)tv.tv_sec,
                   (long)tv.tv_usec);
    }
    if (siocgstampns) {
        if (ioctl(sock, SIOCGSTAMPNS, &ts))
            printf("   %s: %s\n", "SIOCGSTAMPNS", strerror(errno));
        else
            printf("SIOCGSTAMPNS %ld.%09ld\n",
                   (long)ts.tv_sec,
                   (long)ts.tv_nsec);
    }
}

static void recvpacket(int sock, int recvmsg_flags,
               int siocgstamp, int siocgstampns)
{
    char data[256];
    struct msghdr msg;
    struct iovec entry;
    struct sockaddr_in from_addr;
    struct {
        struct cmsghdr cm;
        char control[512];
    } control;
    int res;

    memset(&msg, 0, sizeof(msg));
    msg.msg_iov = &entry;
    msg.msg_iovlen = 1;
    entry.iov_base = data;
    entry.iov_len = sizeof(data);
    msg.msg_name = (caddr_t)&from_addr;
    msg.msg_namelen = sizeof(from_addr);
    msg.msg_control = &control;
    msg.msg_controllen = sizeof(control);

    res = recvmsg(sock, &msg, recvmsg_flags|MSG_DONTWAIT);
    if (res < 0) {
        printf("%s %s: %s\n",
               "recvmsg",
               (recvmsg_flags & MSG_ERRQUEUE) ? "error" : "regular",
               strerror(errno));
    } else {
        printpacket(&msg, res, data,
                sock, recvmsg_flags,
                siocgstamp, siocgstampns);
    }
}

int main(int argc, char **argv)
{
    int so_timestamping_flags = 0;
    int so_timestamp = 0;
    int so_timestampns = 0;
    int siocgstamp = 0;
    int siocgstampns = 0;
    int ip_multicast_loop = 0;
    char *interface;
    int i;
    int enabled = 1;
    int sock;
    struct ifreq device;
    struct ifreq hwtstamp;
```

```
struct hwtstamp_config hwconfig, hwconfig_requested;
struct sockaddr_in addr;
struct ip_mreq imr;
struct in_addr iaddr;
int val;
socklen_t len;
struct timeval next;

if (argc < 2)
    usage(0);
interface = argv[1];

for (i = 2; i < argc; i++) {
    if (!strcasecmp(argv[i], "SO_TIMESTAMP"))
        so_timestamp = 1;
    else if (!strcasecmp(argv[i], "SO_TIMESTAMPNS"))
        so_timestampns = 1;
    else if (!strcasecmp(argv[i], "SIOCGSTAMP"))
        siocgstamp = 1;
    else if (!strcasecmp(argv[i], "SIOCGSTAMPNS"))
        siocgstampns = 1;
    else if (!strcasecmp(argv[i], "IP_MULTICAST_LOOP"))
        ip_multicast_loop = 1;
    else if (!strcasecmp(argv[i], "SOF_TIMESTAMPING_TX_HARDWARE"))
        so_timestamping_flags |= SOF_TIMESTAMPING_TX_HARDWARE;
    else if (!strcasecmp(argv[i], "SOF_TIMESTAMPING_TX_SOFTWARE"))
        so_timestamping_flags |= SOF_TIMESTAMPING_TX_SOFTWARE;
    else if (!strcasecmp(argv[i], "SOF_TIMESTAMPING_RX_HARDWARE"))
        so_timestamping_flags |= SOF_TIMESTAMPING_RX_HARDWARE;
    else if (!strcasecmp(argv[i], "SOF_TIMESTAMPING_RX_SOFTWARE"))
        so_timestamping_flags |= SOF_TIMESTAMPING_RX_SOFTWARE;
    else if (!strcasecmp(argv[i], "SOF_TIMESTAMPING_SOFTWARE"))
        so_timestamping_flags |= SOF_TIMESTAMPING_SOFTWARE;
    else if (!strcasecmp(argv[i], "SOF_TIMESTAMPING_SYS_HARDWARE"))
        so_timestamping_flags |= SOF_TIMESTAMPING_SYS_HARDWARE;
    else if (!strcasecmp(argv[i], "SOF_TIMESTAMPING_RAW_HARDWARE"))
        so_timestamping_flags |= SOF_TIMESTAMPING_RAW_HARDWARE;
    else
        usage(argv[i]);
}

sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP);
if (sock < 0)
    bail("socket");

memset(&device, 0, sizeof(device));
strncpy(device.ifr_name, interface, sizeof(device.ifr_name));
if (ioctl(sock, SIOCGIFADDR, &device) < 0)
    bail("getting interface IP address");

memset(&hwtstamp, 0, sizeof(hwtstamp));
strncpy(hwtstamp.ifr_name, interface, sizeof(hwtstamp.ifr_name));
hwtstamp.ifr_data = (void *)&hwconfig;
memset(&hwconfig, 0, sizeof(hwconfig));
hwconfig.tx_type =
    (so_timestamping_flags & SOF_TIMESTAMPING_TX_HARDWARE) ?
    HWTSTAMP_TX_ON : HWTSTAMP_TX_OFF;
hwconfig.rx_filter =
    (so_timestamping_flags & SOF_TIMESTAMPING_RX_HARDWARE) ?
    HWTSTAMP_FILTER_PTP_V1_L4_SYNC : HWTSTAMP_FILTER_NONE;
hwconfig_requested = hwconfig;
if (ioctl(sock, SIOCSHWTSTAMP, &hwtstamp) < 0) {
    if ((errno == EINVAL || errno == ENOTSUP) &&
        hwconfig_requested.tx_type == HWTSTAMP_TX_OFF &&
        hwconfig_requested.rx_filter == HWTSTAMP_FILTER_NONE)
        printf("SIOCSHWTSTAMP: disabling hardware time stamping not possible\n");
```

```
    else
        bail("SIOCSHWTSTAMP");
}
printf("SIOCSHWTSTAMP: tx_type %d requested, got %d; rx_filter %d requested, got %d\n",
       hwconfig_requested.tx_type, hwconfig.tx_type,
       hwconfig_requested.rx_filter, hwconfig.rx_filter);

/* bind to PTP port */
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = htonl(INADDR_ANY);
addr.sin_port = htons(319 /* PTP event port */);
if (bind(sock,
     (struct sockaddr *)&addr,
     sizeof(struct sockaddr_in)) < 0)
    bail("bind");

/* set multicast group for outgoing packets */
inet_aton("224.0.1.130", &iaddr); /* alternate PTP domain 1 */
addr.sin_addr = iaddr;
imr.imr_multiaddr.s_addr = iaddr.s_addr;
imr.imr_interface.s_addr =
    ((struct sockaddr_in *)&device.ifr_addr)->sin_addr.s_addr;
if (setsockopt(sock, IPPROTO_IP, IP_MULTICAST_IF,
           &imr.imr_interface.s_addr, sizeof(struct in_addr)) < 0)
    bail("set multicast");

/* join multicast group, loop our own packet */
if (setsockopt(sock, IPPROTO_IP, IP_ADD_MEMBERSHIP,
           &imr, sizeof(struct ip_mreq)) < 0)
    bail("join multicast group");

if (setsockopt(sock, IPPROTO_IP, IP_MULTICAST_LOOP,
           &ip_multicast_loop, sizeof(enabled)) < 0) {
    bail("loop multicast");
}

/* set socket options for time stamping */
if (so_timestamp &&
    setsockopt(sock, SOL_SOCKET, SO_TIMESTAMP,
           &enabled, sizeof(enabled)) < 0)
    bail("setsockopt SO_TIMESTAMP");

if (so_timestampns &&
    setsockopt(sock, SOL_SOCKET, SO_TIMESTAMPNS,
           &enabled, sizeof(enabled)) < 0)
    bail("setsockopt SO_TIMESTAMPNS");

if (so_timestamping_flags &&
    setsockopt(sock, SOL_SOCKET, SO_TIMESTAMPING,
           &so_timestamping_flags,
           sizeof(so_timestamping_flags)) < 0)
    bail("setsockopt SO_TIMESTAMPING");

/* request IP_PKTINFO for debugging purposes */
if (setsockopt(sock, SOL_IP, IP_PKTINFO,
           &enabled, sizeof(enabled)) < 0)
    printf("%s: %s\n", "setsockopt IP_PKTINFO", strerror(errno));

/* verify socket options */
len = sizeof(val);
if (getsockopt(sock, SOL_SOCKET, SO_TIMESTAMP, &val, &len) < 0)
    printf("%s: %s\n", "getsockopt SO_TIMESTAMP", strerror(errno));
else
    printf("SO_TIMESTAMP %d\n", val);

if (getsockopt(sock, SOL_SOCKET, SO_TIMESTAMPNS, &val, &len) < 0)
```

```
    printf("%s: %s\n", "getsockopt SO_TIMESTAMPNS",
           strerror(errno));
else
    printf("SO_TIMESTAMPNS %d\n", val);

if (getsockopt(sock, SOL_SOCKET, SO_TIMESTAMPING, &val, &len) < 0) {
    printf("%s: %s\n", "getsockopt SO_TIMESTAMPING",
           strerror(errno));
} else {
    printf("SO_TIMESTAMPING %d\n", val);
    if (val != so_timestamping_flags)
        printf("   not the expected value %d\n",
               so_timestamping_flags);
}

/* send packets forever every five seconds */
gettimeofday(&next, 0);
next.tv_sec = (next.tv_sec + 1) / 5 * 5;
next.tv_usec = 0;
while (1) {
    struct timeval now;
    struct timeval delta;
    long delta_us;
    int res;
    fd_set readfs, errorfs;

    gettimeofday(&now, 0);
    delta_us = (long)(next.tv_sec - now.tv_sec) * 1000000 +
        (long)(next.tv_usec - now.tv_usec);
    if (delta_us > 0) {
        /* continue waiting for timeout or data */
        delta.tv_sec = delta_us / 1000000;
        delta.tv_usec = delta_us % 1000000;

        FD_ZERO(&readfs);
        FD_ZERO(&errorfs);
        FD_SET(sock, &readfs);
        FD_SET(sock, &errorfs);
        printf("%ld.%06ld: select %ldus\n",
               (long)now.tv_sec, (long)now.tv_usec,
               delta_us);
        res = select(sock + 1, &readfs, 0, &errorfs, &delta);
        gettimeofday(&now, 0);
        printf("%ld.%06ld: select returned: %d, %s\n",
               (long)now.tv_sec, (long)now.tv_usec,
               res,
               res < 0 ? strerror(errno) : "success");
        if (res > 0) {
            if (FD_ISSET(sock, &readfs))
                printf("ready for reading\n");
            if (FD_ISSET(sock, &errorfs))
                printf("has error\n");
            recvpacket(sock, 0,
                   siocgstamp,
                   siocgstampns);
            recvpacket(sock, MSG_ERRQUEUE,
                   siocgstamp,
                   siocgstampns);
        }
    } else {
        /* write one packet */
        sendpacket(sock,
               (struct sockaddr *)&addr,
               sizeof(addr));
        next.tv_sec += 5;
        continue;
```

```
        }
    }

    return 0;
}
```