```
/* getdelays.c
 *
 * Utility to get per-pid and per-tgid delay accounting statistics
 * Also illustrates usage of the taskstats interface
 *
 * Copyright (C) Shailabh Nagar, IBM Corp. 2005
 * Copyright (C) Balbir Singh, IBM Corp. 2006
 * Copyright (c) Jay Lan, SGI. 2006
 *
 * Compile with
 *      gcc -I/usr/src/linux/include getdelays.c -o getdelays
 */

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
#include <poll.h>
#include <string.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <signal.h>

#include <linux/genetlink.h>
#include <linux/taskstats.h>
#include <linux/cgroupstats.h>

/*
 * Generic macros for dealing with netlink sockets. Might be duplicated
 * elsewhere. It is recommended that commercial grade applications use
 * libnl or libnetlink and use the interfaces provided by the library
 */
#define GENLMSG_DATA(glh)       ((void *)(NLMSG_DATA(glh) + GENL_HDRLEN))
#define GENLMSG_PAYLOAD(glh)    (NLMSG_PAYLOAD(glh, 0) - GENL_HDRLEN)
#define NLA_DATA(na)            ((void *)((char*)(na) + NLA_HDRLEN))
#define NLA_PAYLOAD(len)        (len - NLA_HDRLEN)

#define err(code, fmt, arg...)                  \
        do {                                    \
                fprintf(stderr, fmt, ##arg);    \
                exit(code);                     \
        } while (0)

int done;
int rcvbufsz;
char name[100];
int dbg;
int print_delays;
int print_io_accounting;
int print_task_context_switch_counts;
__u64 stime, utime;

#define PRINTF(fmt, arg...) {                   \
            if (dbg) {                          \
			printf(fmt, ##arg);		\
		}					\
	}

/* Maximum size of response requested or message sent */
#define MAX_MSG_SIZE	1024
/* Maximum number of cpus expected to be specified in a cpumask */
#define MAX_CPUS	32

struct msgtemplate {
	struct nlmsghdr n;
	struct genlmsghdr g;
	char buf[MAX_MSG_SIZE];
};

char cpumask[100+6*MAX_CPUS];

static void usage(void)
{
	fprintf(stderr, "getdelays [-dilv] [-w logfile] [-r bufsize] "
			"[-m cpumask] [-t tgid] [-p pid]\n");
	fprintf(stderr, "  -d: print delayacct stats\n");
	fprintf(stderr, "  -i: print IO accounting (works only with -p)\n");
	fprintf(stderr, "  -l: listen forever\n");
	fprintf(stderr, "  -v: debug on\n");
	fprintf(stderr, "  -C: container path\n");
}

/*
 * Create a raw netlink socket and bind
 */
static int create_nl_socket(int protocol)
{
	int fd;
	struct sockaddr_nl local;

	fd = socket(AF_NETLINK, SOCK_RAW, protocol);
	if (fd < 0)
		return -1;

	if (rcvbufsz)
		if (setsockopt(fd, SOL_SOCKET, SO_RCVBUF,
				&rcvbufsz, sizeof(rcvbufsz)) < 0) {
			fprintf(stderr, "Unable to set socket rcv buf size "
					"to %d\n",
				rcvbufsz);
			return -1;
		}

	memset(&local, 0, sizeof(local));
	local.nl_family = AF_NETLINK;

	if (bind(fd, (struct sockaddr *) &local, sizeof(local)) < 0)
		goto error;

	return fd;
```

```
error:
        close(fd);
        return -1;
}


static int send_cmd(int sd, __u16 nlmsg_type, __u32 nlmsg_pid,
             __u8 genl_cmd, __u16 nla_type,
             void *nla_data, int nla_len)
{
        struct nlattr *na;
        struct sockaddr_nl nladdr;
        int r, buflen;
        char *buf;

        struct msgtemplate msg;

        msg.n.nlmsg_len = NLMSG_LENGTH(GENL_HDRLEN);
        msg.n.nlmsg_type = nlmsg_type;
        msg.n.nlmsg_flags = NLM_F_REQUEST;
        msg.n.nlmsg_seq = 0;
        msg.n.nlmsg_pid = nlmsg_pid;
        msg.g.cmd = genl_cmd;
        msg.g.version = 0x1;
        na = (struct nlattr *) GENLMSG_DATA(&msg);
        na->nla_type = nla_type;
        na->nla_len = nla_len + 1 + NLA_HDRLEN;
        memcpy(NLA_DATA(na), nla_data, nla_len);
        msg.n.nlmsg_len += NLMSG_ALIGN(na->nla_len);

        buf = (char *) &msg;
        buflen = msg.n.nlmsg_len ;
        memset(&nladdr, 0, sizeof(nladdr));
        nladdr.nl_family = AF_NETLINK;
        while ((r = sendto(sd, buf, buflen, 0, (struct sockaddr *) &nladdr,
                           sizeof(nladdr))) < buflen) {
                if (r > 0) {
                        buf += r;
                        buflen -= r;
                } else if (errno != EAGAIN)
                        return -1;
        }
        return 0;
}


/*
 * Probe the controller in genetlink to find the family id
 * for the TASKSTATS family
 */
static int get_family_id(int sd)
{
        struct {
                struct nlmsghdr n;
                struct genlmsghdr g;
                char buf[256];
        } ans;

        int id = 0, rc;
        struct nlattr *na;
        int rep_len;

        strcpy(name, TASKSTATS_GENL_NAME);
        rc = send_cmd(sd, GENL_ID_CTRL, getpid(), CTRL_CMD_GETFAMILY,
                        CTRL_ATTR_FAMILY_NAME, (void *)name,
                        strlen(TASKSTATS_GENL_NAME)+1);

        rep_len = recv(sd, &ans, sizeof(ans), 0);
        if (ans.n.nlmsg_type == NLMSG_ERROR ||
            (rep_len < 0) || !NLMSG_OK((&ans.n), rep_len))
                return 0;

        na = (struct nlattr *) GENLMSG_DATA(&ans);
        na = (struct nlattr *) ((char *) na + NLA_ALIGN(na->nla_len));
        if (na->nla_type == CTRL_ATTR_FAMILY_ID) {
                id = *(__u16 *) NLA_DATA(na);
        }
        return id;
}

static void print_delayacct(struct taskstats *t)
{
        printf("\n\nCPU   %15s%15s%15s%15s\n"
               "      %15llu%15llu%15llu%15llu\n"
               "IO    %15s%15s\n"
               "      %15llu%15llu\n"
               "SWAP  %15s%15s\n"
               "      %15llu%15llu\n"
               "RECLAIM  %12s%15s\n"
               "      %15llu%15llu\n",
               "count", "real total", "virtual total", "delay total",
               (unsigned long long)t->cpu_count,
               (unsigned long long)t->cpu_run_real_total,
               (unsigned long long)t->cpu_run_virtual_total,
               (unsigned long long)t->cpu_delay_total,
               "count", "delay total",
               (unsigned long long)t->blkio_count,
               (unsigned long long)t->blkio_delay_total,
               "count", "delay total",
               (unsigned long long)t->swapin_count,
               (unsigned long long)t->swapin_delay_total,
               "count", "delay total",
               (unsigned long long)t->freepages_count,
               (unsigned long long)t->freepages_delay_total);
}

static void task_context_switch_counts(struct taskstats *t)
{
        printf("\n\nTask   %15s%15s\n"
               "       %15llu%15llu\n",
               "voluntary", "nonvoluntary",
               (unsigned long long)t->nvcsw, (unsigned long long)t->nivcsw);
```

```
}

static void print_cgroupstats(struct cgroupstats *c)
{
        printf("sleeping %llu, blocked %llu, running %llu, stopped %llu, "
                "uninterruptible %llu\n", (unsigned long long)c->nr_sleeping,
                (unsigned long long)c->nr_io_wait,
                (unsigned long long)c->nr_running,
                (unsigned long long)c->nr_stopped,
                (unsigned long long)c->nr_uninterruptible);
}


static void print_ioacct(struct taskstats *t)
{
        printf("%s: read=%llu, write=%llu, cancelled_write=%llu\n",
                t->ac_comm,
                (unsigned long long)t->read_bytes,
                (unsigned long long)t->write_bytes,
                (unsigned long long)t->cancelled_write_bytes);
}

int main(int argc, char *argv[])
{
        int c, rc, rep_len, aggr_len, len2;
        int cmd_type = TASKSTATS_CMD_ATTR_UNSPEC;
        __u16 id;
        __u32 mypid;

        struct nlattr *na;
        int nl_sd = -1;
        int len = 0;
        pid_t tid = 0;
        pid_t rtid = 0;

        int fd = 0;
        int count = 0;
        int write_file = 0;
        int maskset = 0;
        char *logfile = NULL;
        int loop = 0;
        int containerset = 0;
        char containerpath[1024];
        int cfd = 0;

        struct msgtemplate msg;

        while (1) {
                c = getopt(argc, argv, "qdiw:r:m:t:p:vlC:");
                if (c < 0)
                        break;

                switch (c) {
                case 'd':
                        printf("print delayacct stats ON\n");
                        print_delays = 1;
```

```
				break;
			case 'i':
				printf("printing IO accounting\n");
				print_io_accounting = 1;
				break;
			case 'q':
				printf("printing task/process context switch rates\n");
				print_task_context_switch_counts = 1;
				break;
			case 'C':
				containerset = 1;
				strncpy(containerpath, optarg, strlen(optarg) + 1);
				break;
			case 'w':
				logfile = strdup(optarg);
				printf("write to file %s\n", logfile);
				write_file = 1;
				break;
			case 'r':
				rcvbufsz = atoi(optarg);
				printf("receive buf size %d\n", rcvbufsz);
				if (rcvbufsz < 0)
					err(1, "Invalid rcv buf size\n");
				break;
			case 'm':
				strncpy(cpumask, optarg, sizeof(cpumask));
				maskset = 1;
				printf("cpumask %s maskset %d\n", cpumask, maskset);
				break;
			case 't':
				tid = atoi(optarg);
				if (!tid)
					err(1, "Invalid tgid\n");
				cmd_type = TASKSTATS_CMD_ATTR_TGID;
				break;
			case 'p':
				tid = atoi(optarg);
				if (!tid)
					err(1, "Invalid pid\n");
				cmd_type = TASKSTATS_CMD_ATTR_PID;
				break;
			case 'v':
				printf("debug on\n");
				dbg = 1;
				break;
			case 'l':
				printf("listen forever\n");
				loop = 1;
				break;
			default:
				usage();
				exit(-1);
			}
		}

		if (write_file) {
```

```
		fd = open(logfile, O_WRONLY | O_CREAT | O_TRUNC,
			  S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH);
		if (fd == -1) {
			perror("Cannot open output file\n");
			exit(1);
		}
	}

	if ((nl_sd = create_nl_socket(NETLINK_GENERIC)) < 0)
		err(1, "error creating Netlink socket\n");


	mypid = getpid();
	id = get_family_id(nl_sd);
	if (!id) {
		fprintf(stderr, "Error getting family id, errno %d\n", errno);
		goto err;
	}
	PRINTF("family id %d\n", id);

	if (maskset) {
		rc = send_cmd(nl_sd, id, mypid, TASKSTATS_CMD_GET,
			      TASKSTATS_CMD_ATTR_REGISTER_CPUMASK,
			      &cpumask, strlen(cpumask) + 1);
		PRINTF("Sent register cpumask, retval %d\n", rc);
		if (rc < 0) {
			fprintf(stderr, "error sending register cpumask\n");
			goto err;
		}
	}

	if (tid && containerset) {
		fprintf(stderr, "Select either -t or -C, not both\n");
		goto err;
	}

	if (tid) {
		rc = send_cmd(nl_sd, id, mypid, TASKSTATS_CMD_GET,
			      cmd_type, &tid, sizeof(__u32));
		PRINTF("Sent pid/tgid, retval %d\n", rc);
		if (rc < 0) {
			fprintf(stderr, "error sending tid/tgid cmd\n");
			goto done;
		}
	}

	if (containerset) {
		cfd = open(containerpath, O_RDONLY);
		if (cfd < 0) {
			perror("error opening container file");
			goto err;
		}
		rc = send_cmd(nl_sd, id, mypid, CGROUPSTATS_CMD_GET,
			      CGROUPSTATS_CMD_ATTR_FD, &cfd, sizeof(__u32));
		if (rc < 0) {
			perror("error sending cgroupstats command");
```

```
				goto err;
			}
		}
		if (!maskset && !tid && !containerset) {
			usage();
			goto err;
		}

		do {
			int i;

			rep_len = recv(nl_sd, &msg, sizeof(msg), 0);
			PRINTF("received %d bytes\n", rep_len);

			if (rep_len < 0) {
				fprintf(stderr, "nonfatal reply error: errno %d\n",
					errno);
				continue;
			}
			if (msg.n.nlmsg_type == NLMSG_ERROR ||
			    !NLMSG_OK((&msg.n), rep_len)) {
				struct nlmsgerr *err = NLMSG_DATA(&msg);
				fprintf(stderr, "fatal reply error,  errno %d\n",
					err->error);
				goto done;
			}

			PRINTF("nlmsghdr size=%zu, nlmsg_len=%d, rep_len=%d\n",
			       sizeof(struct nlmsghdr), msg.n.nlmsg_len, rep_len);


			rep_len = GENLMSG_PAYLOAD(&msg.n);

			na = (struct nlattr *) GENLMSG_DATA(&msg);
			len = 0;
			i = 0;
			while (len < rep_len) {
				len += NLA_ALIGN(na->nla_len);
				switch (na->nla_type) {
				case TASKSTATS_TYPE_AGGR_TGID:
					/* Fall through */
				case TASKSTATS_TYPE_AGGR_PID:
					aggr_len = NLA_PAYLOAD(na->nla_len);
					len2 = 0;
					/* For nested attributes, na follows */
					na = (struct nlattr *) NLA_DATA(na);
					done = 0;
					while (len2 < aggr_len) {
						switch (na->nla_type) {
						case TASKSTATS_TYPE_PID:
							rtid = *(int *) NLA_DATA(na);
							if (print_delays)
								printf("PID\t%d\n",
rtid);
							break;
						case TASKSTATS_TYPE_TGID:
```

```
                                                rtid = *(int *) NLA_DATA(na);
                                                if (print_delays)
                                                        printf("TGID\t%d\n",
rtid);
                                                break;
                                        case TASKSTATS_TYPE_STATS:
                                                count++;
                                                if (print_delays)
                                                        print_delayacct((struct
taskstats *) NLA_DATA(na));
                                                if (print_io_accounting)
                                                        print_ioacct((struct
taskstats *) NLA_DATA(na));
                                                if
(print_task_context_switch_counts)

task_context_switch_counts((struct taskstats *) NLA_DATA(na));
                                                if (fd) {
                                                        if (write(fd,
NLA_DATA(na), na->nla_len) < 0) {
                                                                err(1,"write
error\n");
                                                        }
                                                }
                                                if (!loop)
                                                        goto done;
                                                break;
                                        default:
                                                fprintf(stderr, "Unknown nested"
                                                        " nla_type %d\n",
                                                        na->nla_type);
                                                break;
                                        }
                                        len2 += NLA_ALIGN(na->nla_len);
                                        na = (struct nlattr *) ((char *) na +
len2);
                                }
                                break;

                        case CGROUPSTATS_TYPE_CGROUP_STATS:
                                print_cgroupstats(NLA_DATA(na));
                                break;
                        default:
                                fprintf(stderr, "Unknown nla_type %d\n",
                                        na->nla_type);
                                break;
                        }
                        na = (struct nlattr *) (GENLMSG_DATA(&msg) + len);
                }
        } while (loop);
done:
        if (maskset) {
                rc = send_cmd(nl_sd, id, mypid, TASKSTATS_CMD_GET,
                              TASKSTATS_CMD_ATTR_DEREGISTER_CPUMASK,
                              &cpumask, strlen(cpumask) + 1);
                printf("Sent deregister mask, retval %d\n", rc);
                if (rc < 0)
                        err(rc, "error sending deregister cpumask\n");
        }
err:
        close(nl_sd);
        if (fd)
                close(fd);
        if (cfd)
                close(cfd);
        return 0;
}
```