****
14897 Sat Sep 8 15:26:32 2012
new/usr/src/uts/common/io/pfmod.c
3168 pfmod commands could be more useful
****

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

/*
 * STREAMS Packet Filter Module
 *
 * This module applies a filter to messages arriving on its read
 * queue, passing on messages that the filter accepts adn discarding
 * the others.  It supports ioctls for setting the filter.
 *
 * On the write side, the module simply passes everything through
 * unchanged.
 *
 * Based on SunOS 4.x version.  This version has minor changes:
 *      - general SVR4 porting stuff
 *      - change name and prefixes from "nit" buffer to streams buffer
 *      - multithreading assumes configured as D_MTQPAIR
 */

#include <sys/types.h>
#include <sys/sysmacros.h>
#include <sys/errno.h>
#include <sys/debug.h>
#include <sys/time.h>
#include <sys/stropts.h>
#include <sys/stream.h>
#include <sys/conf.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/kmem.h>
#include <sys/strsun.h>
#include <sys/pfmod.h>
#include <sys/modctl.h>
#include <netinet/in.h>

/*
 * Expanded version of the Packetfilt structure that includes
```

```
 * some additional fields that aid filter execution efficiency.
 */
struct epacketfilt {
        struct Pf_ext_packetfilt        pf;
#define pf_Priority     pf.Pf_Priority
#define pf_FilterLen    pf.Pf_FilterLen
#define pf_Filter       pf.Pf_Filter
        /* pointer to word immediately past end of filter */
        ushort_t                *pf_FilterEnd;
        /* length in bytes of packet prefix the filter examines */
        ushort_t                pf_PByteLen;
};
```
______***unchanged_portion_omitted_***

```
/*
 * Handle write-side M_IOCTL messages.
 */
static void
pfioctl(queue_t *wq, mblk_t *mp)
{
        struct  epacketfilt     *pfp = (struct epacketfilt *)wq->q_ptr;
        struct  Pf_ext_packetfilt       *upfp;
        struct  packetfilt      *opfp;
        ushort_t        *fwp;
        int     arg;
        int     maxoff = 0;
        int     maxoffreg = 0;
        struct iocblk   *iocp = (struct iocblk *)mp->b_rptr;
        int     error;

        switch (iocp->ioc_cmd) {
        case PFIOCSETF:
                /*
                 * Verify argument length. Since the size of packet filter
                 * got increased (ENMAXFILTERS was bumped up to 2047), to
                 * maintain backwards binary compatibility, we need to
                 * check for both possible sizes.
                 */
                switch (iocp->ioc_count) {
                case sizeof (struct Pf_ext_packetfilt):
                        error = miocpullup(mp,
                            sizeof (struct Pf_ext_packetfilt));
                        if (error != 0) {
                                miocnak(wq, mp, 0, error);
                                return;
                        }
                        upfp = (struct Pf_ext_packetfilt *)mp->b_cont->b_rptr;
                        if (upfp->Pf_FilterLen > PF_MAXFILTERS) {
                                miocnak(wq, mp, 0, EINVAL);
                                return;
                        }

                        bcopy(upfp, pfp, sizeof (struct Pf_ext_packetfilt));
                        pfp->pf_FilterEnd = &pfp->pf_Filter[pfp->pf_FilterLen];
                        break;

                case sizeof (struct packetfilt):
                        error = miocpullup(mp, sizeof (struct packetfilt));
                        if (error != 0) {
                                miocnak(wq, mp, 0, error);
                                return;
                        }
                        opfp = (struct packetfilt *)mp->b_cont->b_rptr;
                        /* this strange comparison keeps gcc from complaining */
                        if (opfp->Pf_FilterLen - 1 >= ENMAXFILTERS) {
                                miocnak(wq, mp, 0, EINVAL);
                                return;
                        }

                        pfp->pf.Pf_Priority = opfp->Pf_Priority;
                        pfp->pf.Pf_FilterLen = (unsigned int)opfp->Pf_FilterLen;

                        bcopy(opfp->Pf_Filter, pfp->pf.Pf_Filter,
                            sizeof (opfp->Pf_Filter));
                        pfp->pf_FilterEnd = &pfp->pf_Filter[pfp->pf_FilterLen];
                        break;

                default:
                        miocnak(wq, mp, 0, EINVAL);
                        return;
                }

                /*
                 * Find and record maximum byte offset that the
                 * filter users.  We use this when executing the
                 * filter to determine how much of the packet
                 * body to pull up.  This code depends on the
                 * filter encoding.
                 */
                for (fwp = pfp->pf_Filter; fwp < pfp->pf_FilterEnd; fwp++) {
                        arg = *fwp & ((1 << ENF_NBPA) - 1);
                        switch (arg) {
                        default:
                                if ((arg -= ENF_PUSHWORD) > maxoff)
                                        maxoff = arg;
                                break;

                        case ENF_LOAD_OFFSET:
                                /* Point to the offset */
                                fwp++;
                                if (*fwp > maxoffreg)
                                        maxoffreg = *fwp;
                                break;

                        case ENF_PUSHLIT:
                        case ENF_BRTR:
                        case ENF_BRFL:
                                /* Skip over the literal. */
                                fwp++;
                                break;

                        case ENF_PUSHZERO:
                        case ENF_PUSHONE:
                        case ENF_PUSHFFFF:
                        case ENF_PUSHFF00:
                        case ENF_PUSH00FF:
                        case ENF_PUSHFF00_N:
                        case ENF_PUSH00FF_N:
                        case ENF_NOPUSH:
                        case ENF_POP:
                                break;
                        }
                }

                /*
                 * Convert word offset to length in bytes.
                 */
                pfp->pf_PByteLen = (maxoff + maxoffreg + 1) * sizeof (ushort_t);
                miocack(wq, mp, 0, 0);
                break;

        default:
```

```
                putnext(wq, mp);
                break;
        }
}

/* #define      DEBUG   1 */
/* #define      INNERDEBUG      1 */

#ifdef  INNERDEBUG
#define enprintf(a)     printf a
#else
#define enprintf(a)
#endif

/*
 * Apply the packet filter given by pfp to the packet given by
 * pp.  Return nonzero iff the filter accepts the packet.
 *
 * The packet comes in two pieces, a header and a body, since
 * that's the most convenient form for our caller.  The header
 * is in contiguous memory, whereas the body is in a mbuf.
 * Our caller will have adjusted the mbuf chain so that its first
 * min(MLEN, length(body)) bytes are guaranteed contiguous.  For
 * the sake of efficiency (and some laziness) the filter is prepared
 * to examine only these two contiguous pieces.  Furthermore, it
 * assumes that the header length is even, so that there's no need
 * to glue the last byte of header to the first byte of data.
 */

#define opx(i)  ((i) >> ENF_NBPA)

static int
FilterPacket(struct packdesc *pp, struct epacketfilt *pfp)
{
        int             maxhdr = pp->pd_hdrlen;
        int             maxword = maxhdr + pp->pd_bodylen;
        ushort_t        *sp;
        ushort_t        *fp;
        ushort_t        *fpe;
        unsigned        op;
        unsigned        arg;
        unsigned        offreg = 0;
        ushort_t        stack[ENMAXFILTERS+1];

        fp = &pfp->pf_Filter[0];
        fpe = pfp->pf_FilterEnd;

        enprintf(("FilterPacket(%p, %p, %p, %p):\n", pp, pfp, fp, fpe));

        /*
         * Push TRUE on stack to start.  The stack size is chosen such
         * that overflow can't occur -- each operation can push at most
         * one item on the stack, and the stack size equals the maximum
         * program length.
         */
        sp = &stack[ENMAXFILTERS];
        *sp = 1;

        while (fp < fpe) {
        op = *fp >> ENF_NBPA;
        arg = *fp & ((1 << ENF_NBPA) - 1);
        fp++;

        switch (arg) {
        default:
                arg -= ENF_PUSHWORD;
```

```
		/*
		 * Since arg is unsigned,
		 * if it were less than ENF_PUSHWORD before,
		 * it would now be huge.
		 */
		if (arg + offreg < maxhdr)
			*--sp = pp->pd_hdr[arg + offreg];
		else if (arg + offreg < maxword)
			*--sp = pp->pd_body[arg - maxhdr + offreg];
		else {
			enprintf(("=>0(len)\n"));
			return (0);
		}
		break;
	case ENF_PUSHLIT:
		*--sp = *fp++;
		break;
	case ENF_PUSHZERO:
		*--sp = 0;
		break;
	case ENF_PUSHONE:
		*--sp = 1;
		break;
	case ENF_PUSHFFFF:
		*--sp = 0xffff;
		break;
	case ENF_PUSHFF00:
		*--sp = 0xff00;
		break;
	case ENF_PUSH00FF:
		*--sp = 0x00ff;
		break;
	case ENF_PUSHFF00_N:
		*--sp = htons(0xff00);
		break;
	case ENF_PUSH00FF_N:
		*--sp = htons(0x00ff);
		break;
	case ENF_LOAD_OFFSET:
		offreg = *fp++;
		break;
	case ENF_BRTR:
		if (*sp != 0)
			fp += *fp;
		else
			fp++;
		if (fp >= fpe) {
			enprintf(("BRTR: fp>=fpe\n"));
			return (0);
		}
		break;
	case ENF_BRFL:
		if (*sp == 0)
			fp += *fp;
		else
			fp++;
		if (fp >= fpe) {
			enprintf(("BRFL: fp>=fpe\n"));
			return (0);
		}
		break;
	case ENF_POP:
		++sp;
		if (sp > &stack[ENMAXFILTERS]) {
			enprintf(("stack underflow\n"));
			return (0);
```

```
		}
		break;
	case ENF_NOPUSH:
		break;
	}

	if (sp < &stack[2]) {	/* check stack overflow: small yellow zone */
		enprintf(("=>0(--sp)\n"));
		return (0);
	}

	if (op == ENF_NOP)
		continue;

	/*
	 * all non-NOP operators binary, must have at least two operands
	 * on stack to evaluate.
	 */
	if (sp > &stack[ENMAXFILTERS-2]) {
		enprintf(("=>0(sp++)\n"));
		return (0);
	}

	arg = *sp++;
	switch (op) {
	default:
		enprintf(("=>0(def)\n"));
		return (0);
	case opx(ENF_AND):
		*sp &= arg;
		break;
	case opx(ENF_OR):
		*sp |= arg;
		break;
	case opx(ENF_XOR):
		*sp ^= arg;
		break;
	case opx(ENF_EQ):
		*sp = (*sp == arg);
		break;
	case opx(ENF_NEQ):
		*sp = (*sp != arg);
		break;
	case opx(ENF_LT):
		*sp = (*sp < arg);
		break;
	case opx(ENF_LE):
		*sp = (*sp <= arg);
		break;
	case opx(ENF_GT):
		*sp = (*sp > arg);
		break;
	case opx(ENF_GE):
		*sp = (*sp >= arg);
		break;

	/* short-circuit operators */

	case opx(ENF_COR):
		if (*sp++ == arg) {
			enprintf(("=>COR %x\n", *sp));
			return (1);
		}
		break;
	case opx(ENF_CAND):
		if (*sp++ != arg) {
```

```
                        enprintf(("=>CAND %x\n", *sp));
                        return (0);
                }
                break;
        case opx(ENF_CNOR):
                if (*sp++ == arg) {
                        enprintf(("=>COR %x\n", *sp));
                        return (0);
                }
                break;
        case opx(ENF_CNAND):
                if (*sp++ != arg) {
                        enprintf(("=>CNAND %x\n", *sp));
                        return (1);
                }
                break;
        }
        }
        enprintf(("=>%x\n", *sp));
        return (*sp);
}
```
______***unchanged_portion_omitted_***

```
**********************************************************
    4500 Sat Sep  8 15:26:33 2012
new/usr/src/uts/common/sys/pfmod.h
3168 pfmod commands could be more useful
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _SYS_PFMOD_H
#define _SYS_PFMOD_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * Ioctls.
 */
#define PFIOC           ('P' << 8)
#define PFIOCSETF       (PFIOC|1)       /* replace current packet filter */

#define ENMAXFILTERS    255             /* maximum filter short words */
#define PF_MAXFILTERS   2047            /* max short words for newpacketfilt */

/*
 * filter structure for SETF
 */
struct packetfilt {
        uchar_t Pf_Priority;                    /* priority of filter */
        uchar_t Pf_FilterLen;                   /* length of filter cmd list */
        ushort_t Pf_Filter[ENMAXFILTERS];       /* filter command list */
};
______unchanged_portion_omitted_

/*
 * We now allow specification of up to MAXFILTERS (short) words of a filter
 * command list to be applied to incoming packets to determine if
 * those packets should be given to a particular open ethernet file.
 * Alternatively, PF_MAXFILTERS and Pf_ext_packetfilt structure can be
 * used in case even bigger filter command list is needed.
 *
 * In this context, "word" means a short (16-bit) integer.
```

```
 *
 * The filter command list is specified using ioctl().  Each filter command
 * list specifies a sequence of actions that leaves a boolean value on the
 * top of an internal stack.  There is also an offset register which is
 * initialized to zero.  Each word of the command list specifies an action
 * from the set {PUSHLIT, PUSHZERO, PUSHWORD+N, LOAD_OFFSET, BRTR, BRFL, POP}
 * (see #defines below for definitions), and a binary operator from the set
 * {EQ, LT, LE, GT, GE, AND, OR, XOR} which operates on the top two elements
 * of the stack and replaces them with its result.  The special action NOPUSH
 * and the special operator NOP can be used to only perform the binary
 * operation or to only push a value on the stack.
 *
 * If the final value of the filter operation is true, then the packet is
 * accepted for the open file which specified the filter.
 */

/* these must sum to sizeof (ushort_t)! */
#define ENF_NBPA        10                      /* # bits / action */
#define ENF_NBPO         6                      /* # bits / operator */

/* binary operators */
#define ENF_NOP         (0 << ENF_NBPA)
#define ENF_EQ          (1 << ENF_NBPA)
#define ENF_LT          (2 << ENF_NBPA)
#define ENF_LE          (3 << ENF_NBPA)
#define ENF_GT          (4 << ENF_NBPA)
#define ENF_GE          (5 << ENF_NBPA)
#define ENF_AND         (6 << ENF_NBPA)
#define ENF_OR          (7 << ENF_NBPA)
#define ENF_XOR         (8 << ENF_NBPA)
#define ENF_COR         (9 << ENF_NBPA)
#define ENF_CAND        (10 << ENF_NBPA)
#define ENF_CNOR        (11 << ENF_NBPA)
#define ENF_CNAND       (12 << ENF_NBPA)
#define ENF_NEQ         (13 << ENF_NBPA)

/* stack actions */
#define ENF_NOPUSH      0
#define ENF_PUSHLIT     1  /* Push the next word on the stack */
#define ENF_PUSHZERO    2  /* Push 0 on the stack */
#define ENF_PUSHONE     3  /* Push 1 on the stack */
#define ENF_PUSHFFFF    4  /* Push 0xffff on the stack */
#define ENF_PUSHFF00    5  /* Push 0xff00 on the stack */
#define ENF_PUSH00FF    6  /* Push 0x00ff on the stack */
#define ENF_LOAD_OFFSET 7  /* Load the next word into the offset register */
#define ENF_BRTR        8  /* Branch if the stack's top element is true */
#define ENF_BRFL        9  /* Branch if the stack's top element is false */
#define ENF_POP         10 /* Pop the top element from the stack */
#define ENF_PUSHFF00_N  11 /* Push 0xff00 in network byte order on the stack */
#define ENF_PUSH00FF_N  12 /* Push 0x00ff in network byte order on the stack */
#define ENF_PUSHWORD    16

#ifdef  __cplusplus
}
______unchanged_portion_omitted_
```