

```
new/usr/src/uts/common/io/pfmod.c
```

```
*****
14897 Sat Sep 8 15:26:32 2012
new/usr/src/uts/common/io/pfmod.c
3168 pfmod commands could be more useful
*****
1 /*
2 * CDDL HEADER START
3 *
4 * The contents of this file are subject to the terms of the
5 * Common Development and Distribution License (the "License").
6 * You may not use this file except in compliance with the License.
7 *
8 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2006 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

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

26 /*
27 * STREAMS Packet Filter Module
28 *
29 * This module applies a filter to messages arriving on its read
30 * queue, passing on messages that the filter accepts and discarding
31 * the others. It supports ioctl's for setting the filter.
32 *
33 * On the write side, the module simply passes everything through
34 * unchanged.
35 *
36 * Based on SunOS 4.x version. This version has minor changes:
37 * - general SVR4 porting stuff
38 * - change name and prefixes from "nit" buffer to streams buffer
39 * - multithreading assumes configured as D_MTQPAIR
40 */

42 #include <sys/types.h>
43 #include <sys/sysmacros.h>
44 #include <sys/errno.h>
45 #include <sys/debug.h>
46 #include <sys/time.h>
47 #include <sys/stropts.h>
48 #include <sys/stream.h>
49 #include <sys/conf.h>
50 #include <sys/ddi.h>
51 #include <sys/sunddi.h>
52 #include <sys/kmem.h>
53 #include <sys/strsun.h>
54 #include <sys/pfmod.h>
55 #include <sys/modctl.h>
56 #include <netinet/in.h>

58 /*
59 * Expanded version of the Packetfilt structure that includes

```

```
1
```

```
new/usr/src/uts/common/io/pfmod.c
```

```
60     * some additional fields that aid filter execution efficiency.
61     */
62 struct epacketfilt {
63     struct Pf_ext_packetfilt     pf;
64 #define pf_Priority      pf.PF_Priority
65 #define pf_FilterLen    pf.PF_FilterLen
66 #define pf_Filter      pf.PF_Filter
67     /* pointer to word immediately past end of filter */
68     ushort_t                    *pf_FilterEnd;
69     /* length in bytes of packet prefix the filter examines */
70     ushort_t                    pf_PByteLen;
71 };

72 unchanged_portion_omitted

334 /*
335 * Handle write-side M_IOCTL messages.
336 */
337 static void
338 pfiocctl(queue_t *wq, mblk_t *mp)
339 {
340     struct epacketfilt    *ppf = (struct epacketfilt *)wq->q_ptr;
341     struct Pf_ext_packetfilt *upfp;
342     struct packetfilt     *opfp;
343     ushort_t              *fwp;
344     int                   arg;
345     int                   maxoff = 0;
346     int                   maxoffreg = 0;
347     struct iocblk         *iocp = (struct iocblk *)mp->b_rptr;
348     int                   error;

350     switch (iocp->ioc_cmd) {
351     case PFIOCSETF:
352         /*
353          * Verify argument length. Since the size of packet filter
354          * got increased (ENMAXFILTERS was bumped up to 2047), to
355          * maintain backwards binary compatibility, we need to
356          * check for both possible sizes.
357         */
358         switch (iocp->ioc_count) {
359             case sizeof (struct Pf_ext_packetfilt):
360                 error = miocpullup(mp,
361                                     sizeof (struct Pf_ext_packetfilt));
362                 if (error != 0) {
363                     miocnak(wq, mp, 0, error);
364                     return;
365                 }
366                 upfp = (struct Pf_ext_packetfilt *)mp->b_cont->b_rptr;
367                 if (upfp->pf_FilterLen > PF_MAXFILTERS) {
368                     miocnak(wq, mp, 0, EINVAL);
369                     return;
370                 }

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

376             case sizeof (struct packetfilt):
377                 error = miocpullup(mp, sizeof (struct packetfilt));
378                 if (error != 0) {
379                     miocnak(wq, mp, 0, error);
380                     return;
381                 }
382                 opfp = (struct packetfilt *)mp->b_cont->b_rptr;
383                 /* this strange comparison keeps gcc from complaining */
384                 if (opfp->pf_FilterLen - 1 >= ENMAXFILTERS) {
385                     miocnak(wq, mp, 0, EINVAL);
386                 }
387             }
388         }
389     }
390 }
```

```
2
```

```

386             return;
387         }
388
389         pfp->pf.Pf_Priority = opfp->Pf_Priority;
390         pfp->pf.Pf_FilterLen = (unsigned int)opfp->Pf_FilterLen;
391
392         bcopy(opfp->Pf_Filter, pfp->pf.Pf_Filter,
393               sizeof (opfp->Pf_Filter));
394         pfp->pf_FilterEnd = &pfp->pf_Filter[pfp->pf_FilterLen];
395         break;
396
397     default:
398         miocnak(wq, mp, 0, EINVAL);
399         return;
400     }
401
402     /*
403      * Find and record maximum byte offset that the
404      * filter users. We use this when executing the
405      * filter to determine how much of the packet
406      * body to pull up. This code depends on the
407      * filter encoding.
408     */
409     for (fwp = pfp->pf_Filter; fwp < pfp->pf_FilterEnd; fwp++) {
410         arg = *fwp & ((1 << ENF_NBPA) - 1);
411         switch (arg) {
412             default:
413                 if ((arg -= ENF_PUSHWORD) > maxoff)
414                     maxoff = arg;
415                 break;
416
417             case ENF_LOAD_OFFSET:
418                 /* Point to the offset */
419                 fwp++;
420                 if (*fwp > maxoffreg)
421                     maxoffreg = *fwp;
422                 break;
423
424             case ENF_PUSHLIT:
425             case ENF_BRTR:
426             case ENF_BRFL:
427                 /* Skip over the literal. */
428                 fwp++;
429                 break;
430
431             case ENF_PUSHZERO:
432             case ENF_PUSHONE:
433             case ENF_PUSHFFFF:
434             case ENF_PUSHFF00:
435             case ENF_PUSH00FF:
436             case ENF_PUSHFF00_N:
437             case ENF_PUSH00FF_N:
438             case ENF_NOPUSH:
439             case ENF_POP:
440                 break;
441             }
442
443             /*
444              * Convert word offset to length in bytes.
445             */
446             pfp->pf_PByteLen = (maxoff + maxoffreg + 1) * sizeof (ushort_t);
447             miocack(wq, mp, 0, 0);
448             break;
449
450     default:

```

```

452             putnext(wq, mp);
453             break;
454         }
455     }
456
457     /* #define      DEBUG      1 */
458     /* #define      INNERDEBUG    1 */
459
460     #ifdef INNERDEBUG
461     #define enprintf(a)      printf a
462     #else
463     #define enprintf(a)
464     #endif
465
466     /*
467      * Apply the packet filter given by pfp to the packet given by
468      * pp. Return nonzero iff the filter accepts the packet.
469      *
470      * The packet comes in two pieces, a header and a body, since
471      * that's the most convenient form for our caller. The header
472      * is in contiguous memory, whereas the body is in a mbuf.
473      * Our caller will have adjusted the mbuf chain so that its first
474      * min(MLEN, length(body)) bytes are guaranteed contiguous. For
475      * the sake of efficiency (and some laziness) the filter is prepared
476      * to examine only these two contiguous pieces. Furthermore, it
477      * assumes that the header length is even, so that there's no need
478      * to glue the last byte of header to the first byte of data.
479     */
480
481     #define opx(i)  ((i) >> ENF_NBPA)
482
483     static int
484     FilterPacket(struct packdesc *pp, struct epacketfilt *pfp)
485     {
486         int          maxhdr = pp->pd_hdrlen;
487         int          maxword = maxhdr + pp->pd_bodylen;
488         ushort_t    *sp;
489         ushort_t    *fp;
490         ushort_t    *fpe;
491         unsigned    op;
492         unsigned    arg;
493         unsigned    offreg = 0;
494         ushort_t    stack[ENMAXFILTERS+1];
495
496         fp = &pfp->pf_Filter[0];
497         fpe = pfp->pf_FilterEnd;
498
499         enprintf(("FilterPacket(%p, %p, %p, %p):\n", pp, pfp, fp, fpe));
500
501         /*
502          * Push TRUE on stack to start. The stack size is chosen such
503          * that overflow can't occur -- each operation can push at most
504          * one item on the stack, and the stack size equals the maximum
505          * program length.
506         */
507         sp = &stack[ENMAXFILTERS];
508         *sp = 1;
509
510         while (fp < fpe) {
511             op = *fp >> ENF_NBPA;
512             arg = *fp & ((1 << ENF_NBPA) - 1);
513             fp++;
514
515             switch (arg) {
516                 default:
517                     arg -= ENF_PUSHWORD;

```

```

518         /*
519          * Since arg is unsigned,
520          * if it were less than ENF_PUSHWORD before,
521          * it would now be huge.
522          */
523         if (arg + offreg < maxhdr)
524             *--sp = pp->pd_hdr[arg + offreg];
525         else if (arg + offreg < maxword)
526             *--sp = pp->pd_body[arg - maxhdr + offreg];
527         else {
528             enprintf(("=>0(len)\n"));
529             return (0);
530         }
531     break;
532 case ENF_PUSHLIT:
533     *--sp = *fp++;
534     break;
535 case ENF_PUSHZERO:
536     *--sp = 0;
537     break;
538 case ENF_PUSHONE:
539     *--sp = 1;
540     break;
541 case ENF_PUSHFFFF:
542     *--sp = 0xffff;
543     break;
544 case ENF_PUSHFF00:
545     *--sp = 0xff00;
546     break;
547 case ENF_PUSH00FF:
548     *--sp = 0x00ff;
549     break;
550 case ENF_PUSHFF00_N:
551     *--sp = htons(0xff00);
552     break;
553 case ENF_PUSH00FF_N:
554     *--sp = htons(0x00ff);
555     break;
556 case ENF_LOAD_OFFSET:
557     offreg = *fp++;
558     break;
559 case ENF_BRTR:
560     if (*sp != 0)
561         fp += *fp;
562     else
563         fp++;
564     if (fp >= fpe) {
565         enprintf(("BRTR: fp>=fpe\n"));
566         return (0);
567     }
568     break;
569 case ENF_BRFL:
570     if (*sp == 0)
571         fp += *fp;
572     else
573         fp++;
574     if (fp >= fpe) {
575         enprintf(("BRFL: fp>=fpe\n"));
576         return (0);
577     }
578     break;
579 case ENF_POP:
580     ++sp;
581     if (sp > &stack[ENMAXFILTERS]) {
582         enprintf(("stack underflow\n"));
583         return (0);

```

```

584         }
585         break;
586     case ENF_NOPUSH:
587         break;
588     }
589     if (sp < &stack[2]) { /* check stack overflow: small yellow zone */
590         enprintf(("=>0(--sp)\n"));
591         return (0);
592     }
593 }
594 if (op == ENF_NOP)
595     continue;
596 /*
597  * all non-NOP operators binary, must have at least two operands
598  * on stack to evaluate.
599  */
600 if (sp > &stack[ENMAXFILTERS-2]) {
601     enprintf(("=>0(sp+)\n"));
602     return (0);
603 }
604 }
605 }
606 arg = *sp++;
607 switch (op) {
608 default:
609     enprintf(("=>0(def)\n"));
610     return (0);
611 case opx(ENF_AND):
612     *sp &= arg;
613     break;
614 case opx(ENF_OR):
615     *sp |= arg;
616     break;
617 case opx(ENF_XOR):
618     *sp ^= arg;
619     break;
620 case opx(ENF_EQ):
621     *sp = (*sp == arg);
622     break;
623 case opx(ENF_NEQ):
624     *sp = (*sp != arg);
625     break;
626 case opx(ENF_LT):
627     *sp = (*sp < arg);
628     break;
629 case opx(ENF_LE):
630     *sp = (*sp <= arg);
631     break;
632 case opx(ENF_GT):
633     *sp = (*sp > arg);
634     break;
635 case opx(ENF_GE):
636     *sp = (*sp >= arg);
637     break;
638 }
639 /*
640  * short-circuit operators */
641 case opx(ENF_COR):
642     if (*sp++ == arg) {
643         enprintf(("=>COR %x\n", *sp));
644         return (1);
645     }
646     break;
647 case opx(ENF_CAND):
648     if (*sp++ != arg) {

```

```
650         enprintf(("%=>CAND %x\n", *sp));
651         return (0);
652     }
653     break;
654 case opx(ENF_C NOR):
655     if (*sp++ == arg) {
656         enprintf(("%=>COR %x\n", *sp));
657         return (0);
658     }
659     break;
660 case opx(ENF_CNAND):
661     if (*sp++ != arg) {
662         enprintf(("%=>CNAND %x\n", *sp));
663         return (1);
664     }
665     break;
666 }
667 }
668 enprintf(("%=>%x\n", *sp));
669 return (*sp);
670 }
```

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
*****
```

```

1 /*
2 * CDDL HEADER START
3 *
4 * The contents of this file are subject to the terms of the
5 * Common Development and Distribution License (the "License").
6 * You may not use this file except in compliance with the License.
7 *
8 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 */
22 * Copyright 2006 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

26 #ifndef _SYS_PFMOD_H
27 #define _SYS_PFMOD_H

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

29 #ifdef __cplusplus
30 extern "C" {
31 #endif

33 /*
34 * Iotcls.
35 */
36 #define PFIOC      ('P' << 8)
37 #define PFIOCSETF   (PFIOC|1)      /* replace current packet filter */

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

42 /*
43 * filter structure for SETF
44 */
45 struct packetfilt {
46     uchar_t Pf_Priority;           /* priority of filter */
47     uchar_t Pf_FilterLen;         /* length of filter cmd list */
48     ushort_t Pf_Filter[ENMAXFILTERS]; /* filter command list */
49 };


---

unchanged portion omitted
```

```

60 /*
61 * We now allow specification of up to MAXFILTERS (short) words of a filter
62 * command list to be applied to incoming packets to determine if
63 * those packets should be given to a particular open ethernet file.
64 * Alternatively, PF_MAXFILTERS and Pf_ext_packetfilt structure can be
65 * used in case even bigger filter command list is needed.
66 *
67 * In this context, "word" means a short (16-bit) integer.
```

```

68 *
69 * The filter command list is specified using ioctl(). Each filter command
70 * list specifies a sequence of actions that leaves a boolean value on the
71 * top of an internal stack. There is also an offset register which is
72 * initialized to zero. Each word of the command list specifies an action
73 * from the set {PUSHLIT, PUSHZERO, PUSHWORD+N, LOAD_OFFSET, BRTR, BRFL, POP}
74 * (see #defines below for definitions), and a binary operator from the set
75 * {EQ, LT, LE, GT, GE, AND, OR, XOR} which operates on the top two elements
76 * of the stack and replaces them with its result. The special action NOPUSH
77 * and the special operator NOP can be used to only perform the binary
78 * operation or to only push a value on the stack.
79 *
80 * If the final value of the filter operation is true, then the packet is
81 * accepted for the open file which specified the filter.
82 */

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

88 /* binary operators */
89 #define ENF_NOP     (0 << ENF_NBPA)
90 #define ENF_EQ      (1 << ENF_NBPA)
91 #define ENF_LT      (2 << ENF_NBPA)
92 #define ENF_LE      (3 << ENF_NBPA)
93 #define ENF_GT      (4 << ENF_NBPA)
94 #define ENF_GE      (5 << ENF_NBPA)
95 #define ENF_AND     (6 << ENF_NBPA)
96 #define ENF_OR      (7 << ENF_NBPA)
97 #define ENF_XOR     (8 << ENF_NBPA)
98 #define ENF_COR     (9 << ENF_NBPA)
99 #define ENF_CAND    (10 << ENF_NBPA)
100 #define ENF_CNOR    (11 << ENF_NBPA)
101 #define ENF_CNAND   (12 << ENF_NBPA)
102 #define ENF_NEQ     (13 << ENF_NBPA)

104 /* stack actions */
105 #define ENF_NOPUSH  0
106 #define ENF_PUSHLIT 1 /* Push the next word on the stack */
107 #define ENF_PUSHZERO 2 /* Push 0 on the stack */
108 #define ENF_PUSHONE  3 /* Push 1 on the stack */
109 #define ENF_PUSHFFFF 4 /* Push 0xffff on the stack */
110 #define ENF_PUSHF00  5 /* Push 0xff00 on the stack */
111 #define ENF_PUSH0OFF 6 /* Push 0x00ff on the stack */
112 #define ENF_LOAD_OFFSET 7 /* Load the next word into the offset register */
113 #define ENF_BRTR    8 /* Branch if the stack's top element is true */
114 #define ENF_BRFL    9 /* Branch if the stack's top element is false */
115 #define ENF_POP     10 /* Pop the top element from the stack */
116 #define ENF_PUSHF00_N 11 /* Push 0xff00 in network byte order on the stack */
117 #define ENF_PUSH0OFF_N 12 /* Push 0x00ff in network byte order on the stack */
118 #define ENF_PUSHWORD 16

120 #ifdef __cplusplus
121 }


---

unchanged portion omitted
```