

```

1  /* Debugging dump procedures for the kernel. */
2
3  #include "inc.h"
4  #include <timers.h>
5  #include <ibm/interrupt.h>
6  #include <minix/endpoint.h>
7  #include <minix/sys_config.h>
8  #include "../kernel/const.h"
9  #include "../kernel/config.h"
10 #include "../kernel/debug.h"
11 #include "../kernel/type.h"
12 #include "../kernel/proc.h"
13 #include "../kernel/ipc.h"
14
15 #define click_to_round_k(n) \
16     ((unsigned) (((unsigned long) (n) << CLICK_SHIFT) + 512) / 1024))
17
18 /* Declare some local dump procedures. */
19 FORWARD _PROTOTYPE( char *proc_name, (int proc_nr)                );
20 FORWARD _PROTOTYPE( char *s_traps_str, (int flags)                );
21 FORWARD _PROTOTYPE( char *s_flags_str, (int flags)                );
22 FORWARD _PROTOTYPE( char *p_rts_flags_str, (int flags)            );
23
24 /* Some global data that is shared among several dumping procedures.
25  * Note that the process table copy has the same name as in the kernel
26  * so that most macros and definitions from proc.h also apply here.
27  */
28 PUBLIC struct proc proc[NR_TASKS + NR_PROCS];
29 PUBLIC struct priv priv[NR_SYS_PROCS];
30 PUBLIC struct boot_image image[NR_BOOT_PROCS];
31
32 /*=====*
33  *                                     timing_dmp                                     *
34  *=====*/
35 PUBLIC void timing_dmp()
36 {
37     #if ! DEBUG_TIME_LOCKS
38         printf("Enable the DEBUG_TIME_LOCKS definition in src/kernel/config.h\n");
39     #else
40         static struct lock_timingdata timingdata[TIMING_CATEGORIES];
41         int r, c, f, skipped = 0, printed = 0, maxlines = 23, x = 0;
42         static int offsetlines = 0;
43
44         if ((r = sys_getlocktimings(&timingdata[0])) != OK) {
45             report("IS", "warning: couldn't get copy of lock timings", r);
46             return;
47         }
48
49         for(c = 0; c < TIMING_CATEGORIES; c++) {
50             int b;
51             if (!timingdata[c].lock_timings_range[0] || !timingdata[c].binsize)
52                 continue;
53             x = printf("%-*s: misses %lu, resets %lu, measurements %lu: ",
54                 TIMING_NAME, timingdata[c].names,
55                 timingdata[c].misses,
56                 timingdata[c].resets,
57                 timingdata[c].measurements);
58             for(b = 0; b < TIMING_POINTS; b++) {
59                 int w;
60                 if (!timingdata[c].lock_timings[b])
61                     continue;
62                 x += (w = printf(" %5d: %5d", timingdata[c].lock_timings_range[0] +
63                     b*timingdata[c].binsize,
64                     timingdata[c].lock_timings[b]));
65                 if (x + w >= 80) { printf("\n"); x = 0; }

```

```

66     }
67     if (x > 0) printf("\n");
68 }
69 #endif
70 }
71
72 /*=====
73 *                                     kmessages_dmp                                     *
74 *=====*/
75 PUBLIC void kmessages_dmp()
76 {
77     struct kmessages kmess;          /* get copy of kernel messages */
78     char print_buf[KMESS_BUF_SIZE+1]; /* this one is used to print */
79     int start;                       /* calculate start of messages */
80     int r;
81
82     /* Try to get a copy of the kernel messages. */
83     if ((r = sys_getkmessages(&kmess)) != OK) {
84         report("IS", "warning: couldn't get copy of kmessages", r);
85         return;
86     }
87
88     /* Try to print the kernel messages. First determine start and copy the
89     * buffer into a print-buffer. This is done because the messages in the
90     * copy may wrap (the kernel buffer is circular).
91     */
92     start = ((kmess.km_next + KMESS_BUF_SIZE) - kmess.km_size) % KMESS_BUF_SIZE;
93     r = 0;
94     while (kmess.km_size > 0) {
95         print_buf[r] = kmess.km_buf[(start+r) % KMESS_BUF_SIZE];
96         r ++;
97         kmess.km_size --;
98     }
99     print_buf[r] = 0;          /* make sure it terminates */
100     printf("Dump of all messages generated by the kernel.\n\n");
101     printf("%s", print_buf);   /* print the messages */
102 }
103
104 /*=====
105 *                                     monparams_dmp                                     *
106 *=====*/
107 PUBLIC void monparams_dmp()
108 {
109     char val[1024];
110     char *e;
111     int r;
112
113     /* Try to get a copy of the boot monitor parameters. */
114     if ((r = sys_getmonparams(val, sizeof(val))) != OK) {
115         report("IS", "warning: couldn't get copy of monitor params", r);
116         return;
117     }
118
119     /* Append new lines to the result. */
120     e = val;
121     do {
122         e += strlen(e);
123         *e++ = '\n';
124     } while (*e != 0);
125
126     /* Finally, print the result. */
127     printf("Dump of kernel environment strings set by boot monitor.\n");
128     printf("\n%s\n", val);
129 }
130

```

```

131  /*=====*
132  *                               irqtab_dmp                               *
133  *=====*/
134  PUBLIC void irqtab_dmp()
135  {
136      int i,r;
137      struct irq_hook irq_hooks[NR_IRQ_HOOKS];
138      int irq_actids[NR_IRQ_VECTORS];
139      struct irq_hook *e; /* irq tab entry */
140
141      if ((r = sys_getirqhooks(irq_hooks)) != OK) {
142          report("IS","warning: couldn't get copy of irq hooks", r);
143          return;
144      }
145      if ((r = sys_getirqactids(irq_actids)) != OK) {
146          report("IS","warning: couldn't get copy of irq mask", r);
147          return;
148      }
149
150      #if 0
151          printf("irq_actids:");
152          for (i= 0; i<NR_IRQ_VECTORS; i++)
153              printf(" [%d] = 0x%08x", i, irq_actids[i]);
154          printf("\n");
155      #endif
156
157      printf("IRQ policies dump shows use of kernel's IRQ hooks.\n");
158      printf("-h.id- -proc.nr- -irq nr- -policy- -notify id-\n");
159      for (i=0; i<NR_IRQ_HOOKS; i++) {
160          e = &irq_hooks[i];
161          printf("%3d", i);
162          if (e->proc_nr_e==NONE) {
163              printf(" <unused>\n");
164              continue;
165          }
166          printf("%10d ", e->proc_nr_e);
167          printf(" (%02d) ", e->irq);
168          printf(" %s", (e->policy & IRQ_REENABLE) ? "reenable" : " - ");
169          printf(" %d", e->notify_id);
170          if (irq_actids[e->irq] & (1 << i))
171              printf("masked");
172          printf("\n");
173      }
174      printf("\n");
175  }
176
177  /*=====*
178  *                               image_dmp                               *
179  *=====*/
180  PUBLIC void image_dmp()
181  {
182      int m, i,j,r;
183      struct boot_image *ip;
184      static char ipc_to[BITCHUNK_BITS*2];
185
186      if ((r = sys_getimage(image)) != OK) {
187          report("IS","warning: couldn't get copy of image table", r);
188          return;
189      }
190      printf("Image table dump showing all processes included in system image.\n");
191      printf("---name-- -nr- -flags- -traps- -sq- ----pc- -stack- -ipc_to[0]-----\n");
192      for (m=0; m<NR_BOOT_PROCS; m++) {
193          ip = &image[m];
194          for (i=j=0; i < BITCHUNK_BITS; i++, j++) {
195              ipc_to[j] = (ip->ipc_to & (1<<i)) ? '1' : '0';

```

```

196         if (i % 8 == 7) ipc_to[++j] = ' ';
197     }
198     ipc_to[j] = '\\0';
199     printf("%8s %4d  %s  %s  %3d %7lu %7lu  %s\\n",
200         ip->proc_name, ip->proc_nr,
201         s_flags_str(ip->flags), s_traps_str(ip->trap_mask),
202         ip->priority, (long)ip->initial_pc, ip->stksize, ipc_to);
203 }
204 printf("\\n");
205 }
206
207 /*=====
208 *                                     sched_dmp                                     *
209 *=====*/
210 PUBLIC void sched_dmp()
211 {
212     struct proc *rdy_head[NR_SCHED_QUEUES];
213     struct kinfo kinfo;
214     register struct proc *rp;
215     vir_bytes ptr_diff;
216     int r;
217
218     /* First obtain a scheduling information. */
219     if ((r = sys_getschedinfo(proc, rdy_head)) != OK) {
220         report("IS", "warning: couldn't get copy of process table", r);
221         return;
222     }
223     /* Then obtain kernel addresses to correct pointer information. */
224     if ((r = sys_getkinfo(&kinfo)) != OK) {
225         report("IS", "warning: couldn't get kernel addresses", r);
226         return;
227     }
228
229     /* Update all pointers. Nasty pointer algorithmic ... */
230     ptr_diff = (vir_bytes) proc - (vir_bytes) kinfo.proc_addr;
231     for (r=0; r<NR_SCHED_QUEUES; r++)
232         if (rdy_head[r] != NIL_PROC)
233             rdy_head[r] =
234                 (struct proc *)((vir_bytes) rdy_head[r] + ptr_diff);
235     for (rp=BEG_PROC_ADDR; rp < END_PROC_ADDR; rp++)
236         if (rp->p_nextready != NIL_PROC)
237             rp->p_nextready =
238                 (struct proc *)((vir_bytes) rp->p_nextready + ptr_diff);
239
240     /* Now show scheduling queues. */
241     printf("Dumping scheduling queues.\\n");
242
243     for (r=0; r<NR_SCHED_QUEUES; r++) {
244         rp = rdy_head[r];
245         if (!rp) continue;
246         printf("%2d: ", r);
247         while (rp != NIL_PROC) {
248             printf("%3d ", rp->p_nr);
249             rp = rp->p_nextready;
250         }
251         printf("\\n");
252     }
253     printf("\\n");
254 }
255
256 /*=====
257 *                                     kenv_dmp                                     *
258 *=====*/
259 PUBLIC void kenv_dmp()
260 {

```

```

261     struct kinfo kinfo;
262     struct machine machine;
263     int r;
264     if ((r = sys_getkinfo(&kinfo)) != OK) {
265         report("IS","warning: couldn't get copy of kernel info struct", r);
266         return;
267     }
268     if ((r = sys_getmachine(&machine)) != OK) {
269         report("IS","warning: couldn't get copy of kernel machine struct", r);
270         return;
271     }
272
273     printf("Dump of kinfo and machine structures.\n\n");
274     printf("Machine structure:\n");
275     printf("- pc_at:      %3d\n", machine.pc_at);
276     printf("- ps_mca:      %3d\n", machine.ps_mca);
277     printf("- processor:    %3d\n", machine.processor);
278     printf("- vdu_ega:      %3d\n", machine.vdu_ega);
279     printf("- vdu_vga:      %3d\n", machine.vdu_vga);
280     printf("Kernel info structure:\n");
281     printf("- code_base:    %5u\n", kinfo.code_base);
282     printf("- code_size:    %5u\n", kinfo.code_size);
283     printf("- data_base:    %5u\n", kinfo.data_base);
284     printf("- data_size:    %5u\n", kinfo.data_size);
285     printf("- proc_addr:    %5u\n", kinfo.proc_addr);
286     printf("- kmem_base:    %5u\n", kinfo.kmem_base);
287     printf("- kmem_size:    %5u\n", kinfo.kmem_size);
288     printf("- bootdev_base: %5u\n", kinfo.bootdev_base);
289     printf("- bootdev_size: %5u\n", kinfo.bootdev_size);
290     printf("- ramdev_base:  %5u\n", kinfo.ramdev_base);
291     printf("- ramdev_size:  %5u\n", kinfo.ramdev_size);
292     printf("- params_base:  %5u\n", kinfo.params_base);
293     printf("- params_size:  %5u\n", kinfo.params_size);
294     printf("- nr_procs:     %3u\n", kinfo.nr_procs);
295     printf("- nr_tasks:     %3u\n", kinfo.nr_tasks);
296     printf("- release:      %.6s\n", kinfo.release);
297     printf("- version:      %.6s\n", kinfo.version);
298 #if DEBUG_LOCK_CHECK
299     printf("- relocking:    %d\n", kinfo.relocking);
300 #endif
301     printf("\n");
302 }
303
304 PRIVATE char *s_flags_str(int flags)
305 {
306     static char str[10];
307     str[0] = (flags & PREEMPTIBLE) ? 'P' : '-';
308     str[1] = '-';
309     str[2] = (flags & BILLABLE) ? 'B' : '-';
310     str[3] = (flags & SYS_PROC) ? 'S' : '-';
311     str[4] = '-';
312     str[5] = '\0';
313
314     return str;
315 }
316
317 PRIVATE char *s_traps_str(int flags)
318 {
319     static char str[10];
320     str[0] = (flags & (1 << ECHO)) ? 'E' : '-';
321     str[1] = (flags & (1 << SEND)) ? 'S' : '-';
322     str[2] = (flags & (1 << RECEIVE)) ? 'R' : '-';
323     str[3] = (flags & (1 << SENDREC)) ? 'B' : '-';
324     str[4] = (flags & (1 << NOTIFY)) ? 'N' : '-';
325     str[5] = '\0';

```

```

326
327     return str;
328 }
329
330 /*=====
331 *                               privileges_dmp                               *
332 *=====*/
333 PUBLIC void privileges_dmp()
334 {
335     register struct proc *rp;
336     static struct proc *oldrp = BEG_PROC_ADDR;
337     register struct priv *sp;
338     int r, i, n = 0;
339
340     /* First obtain a fresh copy of the current process and system table. */
341     if ((r = sys_getprivtab(priv)) != OK) {
342         report("IS", "warning: couldn't get copy of system privileges table", r);
343         return;
344     }
345     if ((r = sys_getproctab(proc)) != OK) {
346         report("IS", "warning: couldn't get copy of process table", r);
347         return;
348     }
349
350     printf("\n--nr-id-name---- -flags- -traps- grants -ipc_to-- -system calls--\n");
351
352     for (rp = oldrp; rp < END_PROC_ADDR; rp++) {
353         if (isemptyp(rp)) continue;
354         if (++n > 23) break;
355         if (proc_nr(rp) == IDLE)         printf("(%2d) ", proc_nr(rp));
356         else if (proc_nr(rp) < 0)        printf("[%2d] ", proc_nr(rp));
357         else                             printf(" %2d ", proc_nr(rp));
358         r = -1;
359         for (sp = &priv[0]; sp < &priv[NR_SYS_PROCS]; sp++)
360             if (sp->s_proc_nr == rp->p_nr) { r++; break; }
361         if (r == -1 && ! (rp->p_rts_flags & SLOT_FREE)) {
362             sp = &priv[USER_PRIV_ID];
363         }
364         printf("(%02u) %-7.7s %s    %s %7d",
365             sp->s_id, rp->p_name,
366             s_flags_str(sp->s_flags), s_traps_str(sp->s_trap_mask),
367             sp->s_grant_entries);
368         for (i=0; i < NR_SYS_PROCS; i += BITCHUNK_BITS) {
369             printf(" %04x", get_sys_bits(sp->s_ipc_to, i));
370         }
371
372         printf(" ");
373         for (i=0; i < NR_SYS_CALLS; i += BITCHUNK_BITS) {
374             printf(" %04x", sp->s_k_call_mask[i/BITCHUNK_BITS]);
375         }
376         printf("\n");
377     }
378
379     if (rp == END_PROC_ADDR) rp = BEG_PROC_ADDR; else printf("--more--\r");
380     oldrp = rp;
381 }
382
383 /*=====
384 *                               messaging_dmp                               *
385 *=====*/
386 PUBLIC void messaging_dmp()
387 {
388     /* Messaging grid dump */
389
390

```

```

391 register struct proc *rrp, *crp;
392 static struct proc *oldrrp = BEG_PROC_ADDR, *oldcrp = BEG_PROC_ADDR;
393 int r, row, col;
394
395 /* First obtain a fresh copy of the current process table. */
396 if ((r = sys_getproctab(proc)) != OK) {
397     report("IS", "warning: couldn't get copy of process table", r);
398     return;
399 }
400
401 printf("\nNumber of messages sent from process in given row to process in given
column:\n");
402
403 /* iterate through the proc table (rows of matrix) */
404 for (rrp = oldrrp, row = 0; rrp < END_PROC_ADDR; rrp++) {
405     if (isemptytyp(rrp)) continue;
406     if (++row > 23) break;
407
408     /* iterate through the message array (columns of matrix) */
409     for (crp = oldcrp, col = 0; crp < END_PROC_ADDR; crp++) {
410         if (isemptytyp(crp)) continue;
411         if (++col > 7) break;
412         if (row == 1) {
413             if (col == 1)
414                 printf("process names|");
415             else {
416                 if (strcmp(crp->p_name, "<unset>"))
417                     printf("%10s|", crp->p_name);
418                 else
419                     printf(" <pid> %3d|", crp->p_nr);
420             }
421         }
422         else {
423             if (col == 1) {
424                 if (strcmp(rrp->p_name, "<unset>"))
425                     printf("%13s|", rrp->p_name);
426                 else
427                     printf("      <pid> %3d|", rrp->p_nr);
428             }
429             else
430                 printf("%10u|", rrp->p_mess_sent[crp->p_nr +
NR_TASKS]);
431         }
432     }
433     printf("\n");
434 }
435
436 /* handle the paging logic */
437 #define LTR_PAGING 1
438 #if LTR_PAGING
439     /* left-to-right, top-to-bottom paging */
440     if (crp == END_PROC_ADDR) {
441         crp = BEG_PROC_ADDR;
442         if (rrp == END_PROC_ADDR)
443             rrp = BEG_PROC_ADDR;
444         else
445             printf("--more-- <Û\r");
446     }
447     else {
448         rrp = oldrrp;
449         printf("--more-- >>\r");
450     }
451 #else
452     /* top-to-bottom, left-to-right paging */
453     if (rrp == END_PROC_ADDR) {
454         rrp = BEG_PROC_ADDR;
455         if (crp == END_PROC_ADDR)
456             crp = BEG_PROC_ADDR;
457         else
458             printf("--more-- @>\r");
459     }
460     else
461         printf("\n");
462 #endif
463 }
464
465 #endif
466
467 #endif
468
469 #endif
470
471 #endif
472
473 #endif
474
475 #endif
476
477 #endif
478
479 #endif
480
481 #endif
482
483 #endif
484
485 #endif
486
487 #endif
488
489 #endif
490
491 #endif
492
493 #endif
494
495 #endif
496
497 #endif
498
499 #endif
500
501 #endif
502
503 #endif
504
505 #endif
506
507 #endif
508
509 #endif
510
511 #endif
512
513 #endif
514
515 #endif
516
517 #endif
518
519 #endif
520
521 #endif
522
523 #endif
524
525 #endif
526
527 #endif
528
529 #endif
530
531 #endif
532
533 #endif
534
535 #endif
536
537 #endif
538
539 #endif
540
541 #endif
542
543 #endif
544
545 #endif
546
547 #endif
548
549 #endif
550
551 #endif
552
553 #endif
554
555 #endif
556
557 #endif
558
559 #endif
560
561 #endif
562
563 #endif
564
565 #endif
566
567 #endif
568
569 #endif
570
571 #endif
572
573 #endif
574
575 #endif
576
577 #endif
578
579 #endif
580
581 #endif
582
583 #endif
584
585 #endif
586
587 #endif
588
589 #endif
590
591 #endif
592
593 #endif
594
595 #endif
596
597 #endif
598
599 #endif
600
601 #endif
602
603 #endif
604
605 #endif
606
607 #endif
608
609 #endif
610
611 #endif
612
613 #endif
614
615 #endif
616
617 #endif
618
619 #endif
620
621 #endif
622
623 #endif
624
625 #endif
626
627 #endif
628
629 #endif
630
631 #endif
632
633 #endif
634
635 #endif
636
637 #endif
638
639 #endif
640
641 #endif
642
643 #endif
644
645 #endif
646
647 #endif
648
649 #endif
650
651 #endif
652
653 #endif
654
655 #endif
656
657 #endif
658
659 #endif
660
661 #endif
662
663 #endif
664
665 #endif
666
667 #endif
668
669 #endif
670
671 #endif
672
673 #endif
674
675 #endif
676
677 #endif
678
679 #endif
680
681 #endif
682
683 #endif
684
685 #endif
686
687 #endif
688
689 #endif
690
691 #endif
692
693 #endif
694
695 #endif
696
697 #endif
698
699 #endif
700
701 #endif
702
703 #endif
704
705 #endif
706
707 #endif
708
709 #endif
710
711 #endif
712
713 #endif
714
715 #endif
716
717 #endif
718
719 #endif
720
721 #endif
722
723 #endif
724
725 #endif
726
727 #endif
728
729 #endif
730
731 #endif
732
733 #endif
734
735 #endif
736
737 #endif
738
739 #endif
740
741 #endif
742
743 #endif
744
745 #endif
746
747 #endif
748
749 #endif
750
751 #endif
752
753 #endif
754
755 #endif
756
757 #endif
758
759 #endif
760
761 #endif
762
763 #endif
764
765 #endif
766
767 #endif
768
769 #endif
770
771 #endif
772
773 #endif
774
775 #endif
776
777 #endif
778
779 #endif
780
781 #endif
782
783 #endif
784
785 #endif
786
787 #endif
788
789 #endif
790
791 #endif
792
793 #endif
794
795 #endif
796
797 #endif
798
799 #endif
800
801 #endif
802
803 #endif
804
805 #endif
806
807 #endif
808
809 #endif
810
811 #endif
812
813 #endif
814
815 #endif
816
817 #endif
818
819 #endif
820
821 #endif
822
823 #endif
824
825 #endif
826
827 #endif
828
829 #endif
830
831 #endif
832
833 #endif
834
835 #endif
836
837 #endif
838
839 #endif
840
841 #endif
842
843 #endif
844
845 #endif
846
847 #endif
848
849 #endif
850
851 #endif
852
853 #endif
854
855 #endif
856
857 #endif
858
859 #endif
860
861 #endif
862
863 #endif
864
865 #endif
866
867 #endif
868
869 #endif
870
871 #endif
872
873 #endif
874
875 #endif
876
877 #endif
878
879 #endif
880
881 #endif
882
883 #endif
884
885 #endif
886
887 #endif
888
889 #endif
890
891 #endif
892
893 #endif
894
895 #endif
896
897 #endif
898
899 #endif
900
901 #endif
902
903 #endif
904
905 #endif
906
907 #endif
908
909 #endif
910
911 #endif
912
913 #endif
914
915 #endif
916
917 #endif
918
919 #endif
920
921 #endif
922
923 #endif
924
925 #endif
926
927 #endif
928
929 #endif
930
931 #endif
932
933 #endif
934
935 #endif
936
937 #endif
938
939 #endif
940
941 #endif
942
943 #endif
944
945 #endif
946
947 #endif
948
949 #endif
950
951 #endif
952
953 #endif
954
955 #endif
956
957 #endif
958
959 #endif
960
961 #endif
962
963 #endif
964
965 #endif
966
967 #endif
968
969 #endif
970
971 #endif
972
973 #endif
974
975 #endif
976
977 #endif
978
979 #endif
980
981 #endif
982
983 #endif
984
985 #endif
986
987 #endif
988
989 #endif
990
991 #endif
992
993 #endif
994
995 #endif
996
997 #endif
998
999 #endif
1000
1001 #endif
1002
1003 #endif
1004
1005 #endif
1006
1007 #endif
1008
1009 #endif
1010
1011 #endif
1012
1013 #endif
1014
1015 #endif
1016
1017 #endif
1018
1019 #endif
1020
1021 #endif
1022
1023 #endif
1024
1025 #endif
1026
1027 #endif
1028
1029 #endif
1030
1031 #endif
1032
1033 #endif
1034
1035 #endif
1036
1037 #endif
1038
1039 #endif
1040
1041 #endif
1042
1043 #endif
1044
1045 #endif
1046
1047 #endif
1048
1049 #endif
1050
1051 #endif
1052
1053 #endif
1054
1055 #endif
1056
1057 #endif
1058
1059 #endif
1060
1061 #endif
1062
1063 #endif
1064
1065 #endif
1066
1067 #endif
1068
1069 #endif
1070
1071 #endif
1072
1073 #endif
1074
1075 #endif
1076
1077 #endif
1078
1079 #endif
1080
1081 #endif
108
```

```

455     }
456     else {
457         crp = oldcrp;
458         printf("--more-- vv\r");
459     }
460 #endif
461     oldcrp = crp;
462     oldrrp = rrp;
463 }
464
465 /*=====
466 *                                     sendmask_dmp                                     *
467 *=====*/
468 PUBLIC void sendmask_dmp()
469 {
470     register struct proc *rp;
471     static struct proc *oldrp = BEG_PROC_ADDR;
472     int r, i, j, n = 0;
473
474     /* First obtain a fresh copy of the current process table. */
475     if ((r = sys_getproctab(proc)) != OK) {
476         report("IS", "warning: couldn't get copy of process table", r);
477         return;
478     }
479
480     printf("\n\n");
481     printf("Sendmask dump for process table. User processes (*) don't have [].");
482     printf("\n");
483     printf("The rows of bits indicate to which processes each process may send.");
484     printf("\n\n");
485
486     #if DEAD_CODE
487         printf(" ");
488         for (j=proc_nr(BEG_PROC_ADDR); j< INIT_PROC_NR+1; j++) {
489             printf("%3d", j);
490         }
491         printf(" *\n");
492
493         for (rp = oldrp; rp < END_PROC_ADDR; rp++) {
494             if (isemptyp(rp)) continue;
495             if (++n > 20) break;
496
497             printf("%8s ", rp->p_name);
498             if (proc_nr(rp) == IDLE)         printf("(%2d ", proc_nr(rp));
499             else if (proc_nr(rp) < 0)        printf("[%2d] ", proc_nr(rp));
500             else                             printf(" %2d ", proc_nr(rp));
501
502             for (j=proc_nr(BEG_PROC_ADDR); j<INIT_PROC_NR+2; j++) {
503                 if (isallowed(rp->p_sendmask, j)) printf(" 1 ");
504                 else                             printf(" 0 ");
505             }
506             printf("\n");
507         }
508         if (rp == END_PROC_ADDR) { printf("\n"); rp = BEG_PROC_ADDR; }
509         else printf("--more--\r");
510         oldrp = rp;
511     #endif
512 }
513
514 PRIVATE char *p_rts_flags_str(int flags)
515 {
516     static char str[10];
517     str[0] = (flags & NO_PRIORITY) ? 's' : '-';
518     str[1] = (flags & SENDING) ? 'S' : '-';
519     str[2] = (flags & RECEIVING) ? 'R' : '-';

```



```

520     str[3] = (flags & SIGNALED)    ? 'I' : '-';
521     str[4] = (flags & SIG_PENDING) ? 'P' : '-';
522     str[5] = (flags & P_STOP)      ? 'T' : '-';
523     str[6] = (flags & NO_PRIV) ? 'p' : '-';
524     str[7] = '\0';
525
526     return str;
527 }
528
529 /*=====
530  *                               proctab_dmp                               *
531  *=====*/
532 #if (CHIP == INTEL)
533 PUBLIC void proctab_dmp()
534 {
535     /* Proc table dump */
536
537     register struct proc *rp;
538     static struct proc *oldrp = BEG_PROC_ADDR;
539     int r, n = 0;
540     phys_clicks text, data, size;
541
542     /* First obtain a fresh copy of the current process table. */
543     if ((r = sys_getproctab(proc)) != OK) {
544         report("IS", "warning: couldn't get copy of process table", r);
545         return;
546     }
547
548     printf("\n-nr-----gen---endpoint-name--- -prior-quant- -user----sys----size-rts
549     flags\n");
550
551     for (rp = oldrp; rp < END_PROC_ADDR; rp++) {
552         if (isemptyp(rp)) continue;
553         if (++n > 23) break;
554         text = rp->p_memmap[T].mem_phys;
555         data = rp->p_memmap[D].mem_phys;
556         size = rp->p_memmap[T].mem_len
557             + ((rp->p_memmap[S].mem_phys + rp->p_memmap[S].mem_len) - data);
558         if (proc_nr(rp) == IDLE) printf("(%2d) ", proc_nr(rp));
559         else if (proc_nr(rp) < 0) printf("[%2d] ", proc_nr(rp));
560         else printf(" %2d ", proc_nr(rp));
561         printf(" %5d %10d ", _ENDPOINT_G(rp->p_endpoint), rp->p_endpoint);
562         printf("%-8.8s %02u/%02u %02d/%02u %6lu %6lu %5uK %s",
563             rp->p_name,
564             rp->p_priority, rp->p_max_priority,
565             rp->p_ticks_left, rp->p_quantum_size,
566             rp->p_user_time, rp->p_sys_time,
567             click_to_round_k(size),
568             p_rts_flags_str(rp->p_rts_flags));
569         if (rp->p_rts_flags & (SENDING|RECEIVING)) {
570             printf(" %-7.7s", proc_name(_ENDPOINT_P(rp->p_getfrom_e)));
571         }
572         printf("\n");
573     }
574     if (rp == END_PROC_ADDR) rp = BEG_PROC_ADDR; else printf("--more--\r");
575     oldrp = rp;
576 }
577 #endif
578 /* (CHIP == INTEL) */
579
580 /*=====
581  *                               memmap_dmp                               *
582  *=====*/
583 PUBLIC void memmap_dmp()
584 {
585     register struct proc *rp;

```

```

584 static struct proc *oldrp = proc;
585 int r, n = 0;
586 phys_clicks size;
587
588 /* First obtain a fresh copy of the current process table. */
589 if ((r = sys_getproctab(proc)) != OK) {
590     report("IS", "warning: couldn't get copy of process table", r);
591     return;
592 }
593
594 printf("\n-nr/name--- --pc-- --sp-- -----text----- -----data----- ----stack-----
--size-\n");
595 for (rp = oldrp; rp < END_PROC_ADDR; rp++) {
596     if (isemptyp(rp)) continue;
597     if (++n > 23) break;
598     size = rp->p_memmap[T].mem_len
599           + ((rp->p_memmap[S].mem_phys + rp->p_memmap[S].mem_len)
600             - rp->p_memmap[D].mem_phys);
601     printf("%3d %-7.7s%7lx%7lx %4x %4x %4x %4x %4x %4x %4x %4x %4x %5uK\n",
602           proc_nr(rp),
603           rp->p_name,
604           (unsigned long) rp->p_reg.pc,
605           (unsigned long) rp->p_reg.sp,
606           rp->p_memmap[T].mem_vir, rp->p_memmap[T].mem_phys, rp->p_memmap[T].
607           mem_len,
608           rp->p_memmap[D].mem_vir, rp->p_memmap[D].mem_phys, rp->p_memmap[D].
609           mem_len,
610           rp->p_memmap[S].mem_vir, rp->p_memmap[S].mem_phys, rp->p_memmap[S].
611           mem_len,
612           click_to_round_k(size));
613 }
614 if (rp == END_PROC_ADDR) rp = proc;
615 else printf("--more--\r");
616 oldrp = rp;
617 }
618
619 /*=====
620 *                               proc_name                               *
621 *=====*/
622 PRIVATE char *proc_name(proc_nr)
623 int proc_nr;
624 {
625     if (proc_nr == ANY) return "ANY";
626     return cproc_addr(proc_nr)->p_name;
627 }

```