```
/* Debugging dump procedures for the kernel. */

#include "inc.h"
#include <timers.h>
#include <ibm/interrupt.h>
#include <minix/endpoint.h>
#include <minix/sys_config.h>
#include "../../kernel/const.h"
#include "../../kernel/config.h"
#include "../../kernel/debug.h"
#include "../../kernel/type.h"
#include "../../kernel/proc.h"
#include "../../kernel/ipc.h"

#define click_to_round_k(n) \
        ((unsigned) ((((unsigned long) (n) << CLICK_SHIFT) + 512) / 1024))

/* Declare some local dump procedures. */
FORWARD _PROTOTYPE( char *proc_name, (int proc_nr)             );
FORWARD _PROTOTYPE( char *s_traps_str, (int flags)             );
FORWARD _PROTOTYPE( char *s_flags_str, (int flags)             );
FORWARD _PROTOTYPE( char *p_rts_flags_str, (int flags)         );

/* Some global data that is shared among several dumping procedures.
 * Note that the process table copy has the same name as in the kernel
 * so that most macros and definitions from proc.h also apply here.
 */
PUBLIC struct proc proc[NR_TASKS + NR_PROCS];
PUBLIC struct priv priv[NR_SYS_PROCS];
PUBLIC struct boot_image image[NR_BOOT_PROCS];

/*===========================================================================*
 *                                timing_dmp                                 *
 *===========================================================================*/
PUBLIC void timing_dmp()
{
#if ! DEBUG_TIME_LOCKS
  printf("Enable the DEBUG_TIME_LOCKS definition in src/kernel/config.h\n");
#else
  static struct lock_timingdata timingdata[TIMING_CATEGORIES];
  int r, c, f, skipped = 0, printed = 0, maxlines = 23, x = 0;
  static int offsetlines = 0;

  if ((r = sys_getlocktimings(&timingdata[0])) != OK) {
      report("IS","warning: couldn't get copy of lock timings", r);
      return;
  }

  for(c = 0; c < TIMING_CATEGORIES; c++) {
        int b;
        if (!timingdata[c].lock_timings_range[0] || !timingdata[c].binsize)
                continue;
        x = printf("%-*s: misses %lu, resets %lu, measurements %lu: ",
        TIMING_NAME, timingdata[c].names,
                timingdata[c].misses,
                timingdata[c].resets,
                timingdata[c].measurements);
        for(b = 0; b < TIMING_POINTS; b++) {
                int w;
                if (!timingdata[c].lock_timings[b])
                        continue;
                x += (w = printf(" %5d: %5d", timingdata[c].lock_timings_range[0] +
                        b*timingdata[c].binsize,
                        timingdata[c].lock_timings[b]));
                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\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   /*===========================================================================*
385    *                                messaging_dmp                              *
386    *===========================================================================*/
387   PUBLIC void messaging_dmp()
388   {
389   /* Messaging grid dump */
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 (isemptyp(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 (isemptyp(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 +
431                                           NR_TASKS]);
432                   }
433           }
434           printf("\n");
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)       rrp = BEG_PROC_ADDR;
443           else                            printf("--more-- <Ù\r");
444     }
445     else {
446           rrp = oldrrp;
447           printf("--more-- >>\r");
448     }
449   #else
450     /* top-to-bottom, left-to-right paging */
451     if (rrp == END_PROC_ADDR) {
452           rrp = BEG_PROC_ADDR;
453           if (crp == END_PROC_ADDR)       crp = BEG_PROC_ADDR;
454           else                            printf("--more-- ©>\r");
```

```
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
      flags\n");
549
550     for (rp = oldrp; rp < END_PROC_ADDR; rp++) {
551           if (isemptyp(rp)) continue;
552           if (++n > 23) break;
553           text = rp->p_memmap[T].mem_phys;
554           data = rp->p_memmap[D].mem_phys;
555           size = rp->p_memmap[T].mem_len
556                   + ((rp->p_memmap[S].mem_phys + rp->p_memmap[S].mem_len) - data);
557           if (proc_nr(rp) == IDLE)        printf("(%2d) ", proc_nr(rp));
558           else if (proc_nr(rp) < 0)       printf("[%2d] ", proc_nr(rp));
559           else                            printf(" %2d  ", proc_nr(rp));
560           printf(" %5d %10d ", _ENDPOINT_G(rp->p_endpoint), rp->p_endpoint);
561           printf("%-8.8s %02u/%02u %02d/%02u %6lu %6lu %5uK %s",
562                  rp->p_name,
563                  rp->p_priority, rp->p_max_priority,
564                  rp->p_ticks_left, rp->p_quantum_size,
565                  rp->p_user_time, rp->p_sys_time,
566                  click_to_round_k(size),
567                  p_rts_flags_str(rp->p_rts_flags));
568           if (rp->p_rts_flags & (SENDING|RECEIVING)) {
569                   printf(" %-7.7s", proc_name(_ENDPOINT_P(rp->p_getfrom_e)));
570           }
571           printf("\n");
572     }
573     if (rp == END_PROC_ADDR) rp = BEG_PROC_ADDR; else printf("--more--\r");
574     oldrp = rp;
575   }
576   #endif                          /* (CHIP == INTEL) */
577
578   /*==========================================================================*
579    *                                memmap_dmp                                *
580    *==========================================================================*/
581   PUBLIC void memmap_dmp()
582   {
583     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].
                     mem_len,
607                  rp->p_memmap[D].mem_vir, rp->p_memmap[D].mem_phys, rp->p_memmap[D].
                     mem_len,
608                  rp->p_memmap[S].mem_vir, rp->p_memmap[S].mem_phys, rp->p_memmap[S].
                     mem_len,
609                  click_to_round_k(size));
610     }
611     if (rp == END_PROC_ADDR) rp = proc;
612     else printf("--more--\r");
613     oldrp = rp;
614   }
615
616   /*===========================================================================*
617    *                                proc_name                                  *
618    *===========================================================================*/
619   PRIVATE char *proc_name(proc_nr)
620   int proc_nr;
621   {
622     if (proc_nr == ANY) return "ANY";
623     return cproc_addr(proc_nr)->p_name;
624   }
625
626
```