

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

```
#include "inc.h"
#include <timers.h>
#include <ibm/interrupt.h>
#include <minix/endpoint.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];
```

```
/* CODIGO PRACTICA 3 */
```

```
PUBLIC long msg_sn[NR_TASKS + NR_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; }
}
if (x > 0) printf("\n");
}
#endif
}

/*-----*
*                kmessages_dmp                *
*-----*/
PUBLIC void kmessages_dmp()
{
    struct kmessages kmess;          /* get copy of kernel messages */
    char print_buf[KMESS_BUF_SIZE+1]; /* this one is used to print */
    int start;                       /* calculate start of messages */
    int r;

    /* Try to get a copy of the kernel messages. */
    if ((r = sys_getkmessages(&kmess)) != OK) {
        report("IS", "warning: couldn't get copy of kmessages", r);
        return;
    }

    /* Try to print the kernel messages. First determine start and copy the
    * buffer into a print-buffer. This is done because the messages in the
    * copy may wrap (the kernel buffer is circular).
    */
    start = ((kmess.km_next + KMESS_BUF_SIZE) - kmess.km_size) % KMESS_BUF_SIZE;
    r = 0;
    while (kmess.km_size > 0) {
        print_buf[r] = kmess.km_buf[(start+r) % KMESS_BUF_SIZE];
        r ++;
        kmess.km_size --;
    }
    print_buf[r] = 0;          /* make sure it terminates */
    printf("Dump of all messages generated by the kernel.\n\n");
    printf("%s", print_buf);    /* print the messages */
}

/*-----*
*                monparams_dmp                *
*-----*/
PUBLIC void monparams_dmp()
{
    char val[1024];
    char *e;
    int r;

```

```

/* Try to get a copy of the boot monitor parameters.*/
if ((r = sys_getmonparams(val, sizeof(val))) != OK) {
    report("IS", "warning: couldn't get copy of monitor params", r);
    return;
}

/* Append new lines to the result.*/
e = val;
do {
    e += strlen(e);
    *e++ = '\n';
} while (*e != 0);

/* Finally, print the result.*/
printf("Dump of kernel environment strings set by boot monitor.\n");
printf("\n%s\n", val);
}

/*-----*
*                irqtab_dmp                *
*-----*/
PUBLIC void irqtab_dmp()
{
    int i,r;
    struct irq_hook irq_hooks[NR_IRQ_HOOKS];
    int irq_actids[NR_IRQ_VECTORS];
    struct irq_hook *e;    /* irq tab entry */
    char *irq[] = {
        "clock",    /* 00 */
        "keyboard", /* 01 */
        "cascade",  /* 02 */
        "rs232",    /* 03 */
        "rs232",    /* 04 */
        "NIC(eth)", /* 05 */
        "floppy",   /* 06 */
        "printer",  /* 07 */
        "",         /* 08 */
        "",         /* 09 */
        "",         /* 10 */
        "",         /* 11 */
        "",         /* 12 */
        "",         /* 13 */
        "at_wini_0", /* 14 */
        "at_wini_1", /* 15 */
    };
};

if ((r = sys_getirqhooks(irq_hooks)) != OK) {
    report("IS", "warning: couldn't get copy of irq hooks", r);
    return;
}
if ((r = sys_getirqactids(irq_actids)) != OK) {
    report("IS", "warning: couldn't get copy of irq mask", r);
    return;
}

#if 0
printf("irq_actids:");

```

```

for (i= 0; i<NR_IRQ_VECTORS; i++)
    printf(" [%d] = 0x%08x", i, irq_actids[i]);
printf("\n");
#endif

printf("IRQ policies dump shows use of kernel's IRQ hooks.\n");
printf("-h.id- -proc.nr- -IRQ vector (nr.)- -policy- -notify id-\n");
for (i=0; i<NR_IRQ_HOOKS; i++) {
    e = &irq_hooks[i];
    printf("%3d", i);
    if (e->proc_nr_e==NONE) {
        printf(" <unused>\n");
        continue;
    }
    printf("%10d ", e->proc_nr_e);
    printf(" %9.9s (%02d) ", irq[e->irq], e->irq);
    printf(" %s", (e->policy & IRQ_REENABLE) ? "reenable" : " - ");
    printf(" %d", e->notify_id);
    if (irq_actids[e->irq] & (1 << i))
        printf("masked");
    printf("\n");
}
printf("\n");
}

/*=====
*                image_dmp                *
*=====*/
PUBLIC void image_dmp()
{
    int m, i,j,r;
    struct boot_image *ip;
    static char ipc_to[BITCHUNK_BITS*2];

    if ((r = sys_getimage(image)) != OK) {
        report("IS","warning: couldn't get copy of image table", r);
        return;
    }
    printf("Image table dump showing all processes included in system image.\n");
    printf("----name-- -nr- -flags- -traps- -sq- ----pc- -stack- -ipc_to[0]-----\n");
    for (m=0; m<NR_BOOT_PROCS; m++) {
        ip = &image[m];
        for (i=j=0; i < BITCHUNK_BITS; i++, j++) {
            ipc_to[j] = (ip->ipc_to & (1<<i)) ? '1' : '0';
            if (i % 8 == 7) ipc_to[++j] = ' ';
        }
        ipc_to[j] = '\0';
        printf("%8s %4d %s %s %3d %7lu %7lu %s\n",
            ip->proc_name, ip->proc_nr,
            s_flags_str(ip->flags), s_traps_str(ip->trap_mask),
            ip->priority, (long)ip->initial_pc, ip->stksize, ipc_to);
    }
    printf("\n");
}

/*=====
*                sched_dmp                *
*=====*/

```

```

*=====*/
PUBLIC void sched_dmp()
{
    struct proc *rdy_head[NR_SCHED_QUEUES];
    struct kinfo kinfo;
    register struct proc *rp;
    vir_bytes ptr_diff;
    int r;

    /* First obtain a scheduling information. */
    if ((r = sys_getschedinfo(proc, rdy_head)) != OK) {
        report("IS", "warning: couldn't get copy of process table", r);
        return;
    }
    /* Then obtain kernel addresses to correct pointer information. */
    if ((r = sys_getkinfo(&kinfo)) != OK) {
        report("IS", "warning: couldn't get kernel addresses", r);
        return;
    }

    /* Update all pointers. Nasty pointer algorithmic ... */
    ptr_diff = (vir_bytes) proc - (vir_bytes) kinfo.proc_addr;
    for (r=0;r<NR_SCHED_QUEUES; r++)
        if (rdy_head[r] != NIL_PROC)
            rdy_head[r] =
                (struct proc *)((vir_bytes) rdy_head[r] + ptr_diff);
    for (rp=BEG_PROC_ADDR; rp < END_PROC_ADDR; rp++)
        if (rp->p_nextready != NIL_PROC)
            rp->p_nextready =
                (struct proc *)((vir_bytes) rp->p_nextready + ptr_diff);

    /* Now show scheduling queues. */
    printf("Dumping scheduling queues.\n");

    for (r=0;r<NR_SCHED_QUEUES; r++) {
        rp = rdy_head[r];
        if (!rp) continue;
        printf("%2d: ", r);
        while (rp != NIL_PROC) {
            printf("%3d ", rp->p_nr);
            rp = rp->p_nextready;
        }
        printf("\n");
    }
    printf("\n");
}

*=====*/
*                kenv_dmp                *
*=====*/
PUBLIC void kenv_dmp()
{
    struct kinfo kinfo;
    struct machine machine;
    int r;
    if ((r = sys_getkinfo(&kinfo)) != OK) {
        report("IS", "warning: couldn't get copy of kernel info struct", r);
    }
}

```

```

    return;
}
if ((r = sys_getmachine(&machine)) != OK) {
    report("IS", "warning: couldn't get copy of kernel machine struct", r);
    return;
}

printf("Dump of kinfo and machine structures.\n\n");
printf("Machine structure:\n");
printf("- pc_at:      %3d\n", machine.pc_at);
printf("- ps_mca:      %3d\n", machine.ps_mca);
printf("- processor:   %3d\n", machine.processor);
printf("- protected:   %3d\n", machine.prot);
printf("- vdu_ega:     %3d\n", machine.vdu_ega);
printf("- vdu_vga:     %3d\n\n", machine.vdu_vga);
printf("Kernel info structure:\n");
printf("- code_base:   %5u\n", kinfo.code_base);
printf("- code_size:   %5u\n", kinfo.code_size);
printf("- data_base:   %5u\n", kinfo.data_base);
printf("- data_size:   %5u\n", kinfo.data_size);
printf("- proc_addr:   %5u\n", kinfo.proc_addr);
printf("- kmem_base:   %5u\n", kinfo.kmem_base);
printf("- kmem_size:   %5u\n", kinfo.kmem_size);
printf("- bootdev_base: %5u\n", kinfo.bootdev_base);
printf("- bootdev_size: %5u\n", kinfo.bootdev_size);
printf("- ramdev_base:  %5u\n", kinfo.ramdev_base);
printf("- ramdev_size:  %5u\n", kinfo.ramdev_size);
printf("- params_base:  %5u\n", kinfo.params_base);
printf("- params_size:  %5u\n", kinfo.params_size);
printf("- nr_procs:     %3u\n", kinfo.nr_procs);
printf("- nr_tasks:     %3u\n", kinfo.nr_tasks);
printf("- release:      %.6s\n", kinfo.release);
printf("- version:      %.6s\n", kinfo.version);
#ifdef DEBUG_LOCK_CHECK
    printf("- relocking:    %d\n", kinfo.relocking);
#endif
printf("\n");
}

PRIVATE char *s_flags_str(int flags)
{
    static char str[10];
    str[0] = (flags & PREEMPTIBLE) ? 'P' : '-';
    str[1] = '-';
    str[2] = (flags & BILLABLE) ? 'B' : '-';
    str[3] = (flags & SYS_PROC) ? 'S' : '-';
    str[4] = '-';
    str[5] = '\0';

    return str;
}

PRIVATE char *s_traps_str(int flags)
{
    static char str[10];
    str[0] = (flags & (1 << ECHO)) ? 'E' : '-';
    str[1] = (flags & (1 << SEND)) ? 'S' : '-';

```

```

str[2] = (flags & (1 << RECEIVE)) ? 'R' : '-';
str[3] = (flags & (1 << SENDREC)) ? 'B' : '-';
str[4] = (flags & (1 << NOTIFY)) ? 'N' : '-';
str[5] = '\0';

return str;
}

/*=====
*                               *
*=====*/
PUBLIC void privileges_dmp()
{
register struct proc *rp;
static struct proc *oldrp = BEG_PROC_ADDR;
register struct priv *sp;
static char ipc_to[NR_SYS_PROCS + 1 + NR_SYS_PROCS/8];
int r, i, j, n = 0;

/* First obtain a fresh copy of the current process and system table. */
if ((r = sys_getprivtab(priv)) != OK) {
report("IS", "warning: couldn't get copy of system privileges table", r);
return;
}
if ((r = sys_getproctab(proc)) != OK) {
report("IS", "warning: couldn't get copy of process table", r);
return;
}

printf("\n--nr-id-name---- -flags- -traps- -ipc_to mask----- \n");

for (rp = oldrp; rp < END_PROC_ADDR; rp++) {
if (isemptyp(rp)) continue;
if (++n > 23) break;
if (proc_nr(rp) == IDLE) printf("(%2d) ", proc_nr(rp));
else if (proc_nr(rp) < 0) printf("[%2d] ", proc_nr(rp));
else printf(" %2d ", proc_nr(rp));
r = -1;
for (sp = &priv[0]; sp < &priv[NR_SYS_PROCS]; sp++)
if (sp->s_proc_nr == rp->p_nr) { r ++; break; }
if (r == -1 && !(rp->p_rts_flags & SLOT_FREE)) {
sp = &priv[USER_PRIV_ID];
}
printf("(%02u) %-7.7s %s %s ",
sp->s_id, rp->p_name,
s_flags_str(sp->s_flags), s_traps_str(sp->s_trap_mask)
);
for (i=j=0; i < NR_SYS_PROCS; i++, j++) {
ipc_to[j] = get_sys_bit(sp->s_ipc_to, i) ? '1' : '0';
if (i % 8 == 7) ipc_to[++j] = ' ';
}
ipc_to[j] = '\0';

printf(" %s \n", ipc_to);
}
if (rp == END_PROC_ADDR) rp = BEG_PROC_ADDR; else printf("--more--\r");
oldrp = rp;

```

```

}

/*-----*
*          sendmask_dmp          *
*-----*/
PUBLIC void sendmask_dmp()
{
    register struct proc *rp;
    static struct proc *oldrp = BEG_PROC_ADDR;
    int r, i, j, n = 0;

    /* First obtain a fresh copy of the current process table. */
    if ((r = sys_getproctab(proc)) != OK) {
        report("IS", "warning: couldn't get copy of process table", r);
        return;
    }

    printf("\n\n");
    printf("Sendmask dump for process table. User processes (*) don't have [].");
    printf("\n");
    printf("The rows of bits indicate to which processes each process may send.");
    printf("\n\n");

#ifdef DEAD_CODE
    printf("          ");
    for (j=proc_nr(BEG_PROC_ADDR); j< INIT_PROC_NR+1; j++) {
        printf("%3d", j);
    }
    printf(" *\n");

    for (rp = oldrp; rp < END_PROC_ADDR; rp++) {
        if (isempty(rp)) continue;
        if (++n > 20) break;

        printf("%8s ", rp->p_name);
        if (proc_nr(rp) == IDLE)    printf("(%2d) ", proc_nr(rp));
        else if (proc_nr(rp) < 0)  printf("[%2d] ", proc_nr(rp));
        else                       printf(" %2d  ", proc_nr(rp));

        for (j=proc_nr(BEG_PROC_ADDR); j<INIT_PROC_NR+2; j++) {
            if (isallowed(rp->p_sendmask, j)) printf(" 1 ");
            else                             printf(" 0 ");
        }
        printf("\n");
    }
    if (rp == END_PROC_ADDR) { printf("\n"); rp = BEG_PROC_ADDR; }
    else printf("--more--\r");
    oldrp = rp;
#endif
}

PRIVATE char *p_rts_flags_str(int flags)
{
    static char str[10];
    str[0] = (flags & NO_MAP) ? 'M' : '-';
    str[1] = (flags & SENDING) ? 'S' : '-';

```

```

str[2] = (flags & RECEIVING)    ? 'R' : '-';
str[3] = (flags & SIGNED)      ? 'I' : '-';
str[4] = (flags & SIG_PENDING)  ? 'P' : '-';
str[5] = (flags & P_STOP)      ? 'T' : '-';
str[6] = '\\0';

return str;
}

/*-----*
 *                *
 *-----*/
#if (CHIP == INTEL)
PUBLIC void proctab_dump()
{
/* Proc table dump */

register struct proc *rp;
static struct proc *oldrp = BEG_PROC_ADDR;
int r, n = 0;
phys_clicks text, data, size;

/* First obtain a fresh copy of the current process table. */
if ((r = sys_getproctab(proc)) != OK) {
report("IS", "warning: couldn't get copy of process table", r);
return;
}

printf("\\n-nr-----gen---endpoint--name--- -prior-quant- -user---sys----size-rts flags-\\n");

for (rp = oldrp; rp < END_PROC_ADDR; rp++) {
if (isemptyp(rp)) continue;
if (++n > 23) break;
text = rp->p_memmap[T].mem_phys;
data = rp->p_memmap[D].mem_phys;
size = rp->p_memmap[T].mem_len
+ ((rp->p_memmap[S].mem_phys + rp->p_memmap[S].mem_len) - data);
if (proc_nr(rp) == IDLE) printf("(%2d) ", proc_nr(rp));
else if (proc_nr(rp) < 0) printf("[%2d] ", proc_nr(rp));
else printf(" %2d ", proc_nr(rp));
printf(" %5d %10d ", _ENDPOINT_G(rp->p_endpoint), rp->p_endpoint);
printf(" %-8.8s %02u/%02u %02d/%02u %6lu%6lu %6uK %s",
rp->p_name,
rp->p_priority, rp->p_max_priority,
rp->p_ticks_left, rp->p_quantum_size,
rp->p_user_time, rp->p_sys_time,
click_to_round_k(size),
p_rts_flags_str(rp->p_rts_flags));
if (rp->p_rts_flags & (SENDING|RECEIVING)) {
printf(" %-7.7s", proc_name(_ENDPOINT_P(rp->p_getfrom_e)));
}
printf("\\n");
}
if (rp == END_PROC_ADDR) rp = BEG_PROC_ADDR; else printf("--more--\\r");
oldrp = rp;
}
#endif /* (CHIP == INTEL) */

```

```

/*=====
*                               *
*                               *
*=====*/
PUBLIC void memmap_dmp()
{
    register struct proc *rp;
    static struct proc *oldrp = proc;
    int r, n = 0;
    phys_clicks size;

    /* First obtain a fresh copy of the current process table. */
    if ((r = sys_getproctab(proc)) != OK) {
        report("IS", "warning: couldn't get copy of process table", r);
        return;
    }

    printf("\n-nr/name--- --pc-- --sp-- -----text----- -----data----- ----stack----- --size-\n"
);
    for (rp = oldrp; rp < END_PROC_ADDR; rp++) {
        if (isempty(rp)) continue;
        if (++n > 23) break;
        size = rp->p_memmap[T].mem_len
            + ((rp->p_memmap[S].mem_phys + rp->p_memmap[S].mem_len)
              - rp->p_memmap[D].mem_phys);
        printf("%3d %-7.7s%7lx%7lx %4x %4x %4x %4x %4x %4x %4x %4x %5uK\n",
            proc_nr(rp),
            rp->p_name,
            (unsigned long) rp->p_reg.pc,
            (unsigned long) rp->p_reg.sp,
            rp->p_memmap[T].mem_vir, rp->p_memmap[T].mem_phys, rp->p_memmap[T].mem_len,
            rp->p_memmap[D].mem_vir, rp->p_memmap[D].mem_phys, rp->p_memmap[D].mem_len,
            rp->p_memmap[S].mem_vir, rp->p_memmap[S].mem_phys, rp->p_memmap[S].mem_len,
            click_to_round_k(size));
    }
    if (rp == END_PROC_ADDR) rp = proc;
    else printf("--more--\r");
    oldrp = rp;
}

/*=====
*                               *
*                               *
*=====*/
PRIVATE char *proc_name(proc_nr)
int proc_nr;
{
    if (proc_nr == ANY) return "ANY";
    return cproc_addr(proc_nr)->p_name;
}

/* CODIGO PRACTICA 3 */
/*=====
*                               *
*                               *
*=====*/
PUBLIC void messgsn_dmp()
{
    int r, i, total = 0;

```

```
/**/ register struct proc *rp;
/**/ static struct proc *oldrp = BEG_PROC_ADDR;

/* Try to get a copy of the MESSG_SN ARRAY.*/
if ((r = sys_getmessgsn(messg_sn)) != OK) {
    report("IS", "warning: couldn't get copy of MESSG_SN ARRAY", r);
    return;
}

/**/ if ((r = sys_getproctab(proc)) != OK) {
/**/     report("IS", "warning: couldn't get copy of PROCESS TABLE", r);
/**/     return;
/**/ }

printf("\n");
printf("nr-name-----sms---- nr-name-----sms---- nr-name-----sms---- nr-name-----sms----\n");
/**/ rp = oldrp;

for (i=0;i<(NR_TASKS+NR_PROCS);i++,rp++) {
    if (isempty(rp)) continue;
    total ++;
    printf("%2d %-7s (%6d) ", i - NR_TASKS, rp->p_name, messg_sn[i] );

    if ((total % 4) == 0)
        printf("\n");
}
printf("\n");
}
```