```
================= kernel/system/do_getinfo.c =============
/* The kernel call implemented in this file:
 *   m_type:    SYS_GETINFO
 *
 * The parameters for this kernel call are:
 *    m1_i3:    I_REQUEST       (what info to get)
 *    m1_p1:    I_VAL_PTR       (where to put it)
 *    m1_i1:    I_VAL_LEN       (maximum length expected, optional)
 *    m1_p2:    I_VAL_PTR2      (second, optional pointer)
 *    m1_i2:    I_VAL_LEN2      (second length or process nr)
 */

#include "../system.h"

static unsigned long bios_buf[1024];    /* 4K, what about alignment */
static vir_bytes bios_buf_vir, bios_buf_len;

#if USE_GETINFO

/*===========================================================================*
 *                                do_getinfo                                 *
 *===========================================================================*/
PUBLIC int do_getinfo(m_ptr)
register message *m_ptr;        /* pointer to request message */
{
/* Request system information to be copied to caller's address space. This
 * call simply copies entire data structures to the caller.
 */
  size_t length;
  phys_bytes src_phys;
  phys_bytes dst_phys;
  int proc_nr, nr;

  /* Set source address and length based on request type. */
  switch (m_ptr->I_REQUEST) {
    case GET_MACHINE: {
        length = sizeof(struct machine);
        src_phys = vir2phys(&machine);
        break;
    }
    case GET_KINFO: {
        length = sizeof(struct kinfo);
        src_phys = vir2phys(&kinfo);
        break;
    }
    case GET_IMAGE: {
        length = sizeof(struct boot_image) * NR_BOOT_PROCS;
        src_phys = vir2phys(image);
        break;
    }
    case GET_IRQHOOKS: {
        length = sizeof(struct irq_hook) * NR_IRQ_HOOKS;
        src_phys = vir2phys(irq_hooks);
        break;
    }
    case GET_SCHEDINFO: {
        /* This is slightly complicated because we need two data structures
         * at once, otherwise the scheduling information may be incorrect.
         * Copy the queue heads and fall through to copy the process table.
         */
        length = sizeof(struct proc *) * NR_SCHED_QUEUES;
        src_phys = vir2phys(rdy_head);
        dst_phys = numap_local(m_ptr->m_source, (vir_bytes) m_ptr->I_VAL_PTR2,
                length);
        if (src_phys == 0 || dst_phys == 0) return(EFAULT);
        phys_copy(src_phys, dst_phys, length);
        /* fall through */
    }
    case GET_PROCTAB: {
        length = sizeof(struct proc) * (NR_PROCS + NR_TASKS);
        src_phys = vir2phys(proc);
        break;
    }
```



```
    case GET_PRIVTAB: {
        length = sizeof(struct priv) * (NR_SYS_PROCS);
        src_phys = vir2phys(priv);
        break;
    }
    case GET_PROC: {
        nr = (m_ptr->I_VAL_LEN2 == SELF) ? m_ptr->m_source : m_ptr->I_VAL_LEN2;
        if (! isokprocn(nr)) return(EINVAL);    /* validate request */
        length = sizeof(struct proc);
        src_phys = vir2phys(proc_addr(nr));
        break;
    }
    case GET_MONPARAMS: {
        src_phys = kinfo.params_base;           /* already is a physical */
        length = kinfo.params_size;
        break;
    }
    case GET_RANDOMNESS: {
        static struct randomness copy;          /* copy to keep counters */
        int i;

        copy = krandom;
        for (i= 0; i<RANDOM_SOURCES; i++) {
                krandom.bin[i].r_size = 0;      /* invalidate random data */
                krandom.bin[i].r_next = 0;
        }
        length = sizeof(struct randomness);
        src_phys = vir2phys(&copy);
        break;
    }
    case GET_KMESSAGES: {
        length = sizeof(struct kmessages);
        src_phys = vir2phys(&kmess);
        break;
    }
#if DEBUG_TIME_LOCKS
    case GET_LOCKTIMING: {
    length = sizeof(timingdata);
    src_phys = vir2phys(timingdata);
    break;
    }
#endif
    case GET_BIOSBUFFER:
        bios_buf_vir = (vir_bytes)bios_buf;
        bios_buf_len = sizeof(bios_buf);

        length = sizeof(bios_buf_len);
        src_phys = vir2phys(&bios_buf_len);
        if (length != m_ptr->I_VAL_LEN2) return (EINVAL);
        proc_nr = m_ptr->m_source;      /* only caller can request copy */
        dst_phys = numap_local(proc_nr, (vir_bytes) m_ptr->I_VAL_PTR2, length);
        if (src_phys == 0 || dst_phys == 0) return(EFAULT);
        phys_copy(src_phys, dst_phys, length);

        length = sizeof(bios_buf_vir);
        src_phys = vir2phys(&bios_buf_vir);
        break;

    default:
        return(EINVAL);
  }

  /* Try to make the actual copy for the requested data. */
  if (m_ptr->I_VAL_LEN > 0 && length > m_ptr->I_VAL_LEN) return (E2BIG);
  proc_nr = m_ptr->m_source;            /* only caller can request copy */
  dst_phys = numap_local(proc_nr, (vir_bytes) m_ptr->I_VAL_PTR, length);
  if (src_phys == 0 || dst_phys == 0) return(EFAULT);
  phys_copy(src_phys, dst_phys, length);
  return(OK);
}

#endif /* USE_GETINFO */
```