```c
/*
 *  linux/drivers/char/mem.c
 *
 *  Copyright (C) 1991, 1992  Linus Torvalds
 */

#include <linux/config.h>
#include <linux/types.h>
#include <linux/errno.h>
#include <linux/sched.h>
#include <linux/kernel.h>
#include <linux/major.h>
#include <linux/tty.h>
#include <linux/miscdevice.h>
#include <linux/tpqic02.h>
#include <linux/ftape.h>
#include <linux/malloc.h>
#include <linux/mman.h>
#include <linux/mm.h>
#include <linux/random.h>

#include <asm/segment.h>
#include <asm/io.h>
#include <asm/pgtable.h>

#ifdef CONFIG_SOUND
void soundcard_init(void);
#endif
#ifdef CONFIG_ISDN
void isdn_init(void);
#endif
#ifdef CONFIG_PCWATCHDOG
void pcwatchdog_init(void);
#endif

static int read_ram(struct inode * inode, struct file * file,
                    char * buf, int count)
{
        return -EIO;
}

static int write_ram(struct inode * inode, struct file * file,
                     const char * buf, int count)
{
        return -EIO;
}

static int read_mem(struct inode * inode, struct file * file, char * buf,
                    int count)
{
        unsigned long p = file->f_pos;
        int read;

        p += PAGE_OFFSET;
        if (count < 0)
                return -EINVAL;
        if (MAP_NR(p) >= MAP_NR(high_memory))
                return 0;
        if (count > high_memory - p)
                count = high_memory - p;
```

```
        read = 0;
        /* we don’t have page 0 mapped on x86/sparc.. */
#if defined(__i386__) || defined(__sparc__)
        while (p < PAGE_OFFSET + PAGE_SIZE && count > 0) {
                put_user(0,buf);
                buf++;
                p++;
                count--;
                read++;
        }
#endif
        memcpy_tofs(buf, (void *) p, count);
        read += count;
        file->f_pos += read;
        return read;
}

static int write_mem(struct inode * inode, struct file * file,
                     const char * buf, int count)
{
        unsigned long p = file->f_pos;
        int written;

        p += PAGE_OFFSET;
        if (count < 0)
                return -EINVAL;
        if (MAP_NR(p) >= MAP_NR(high_memory))
                return 0;
        if (count > high_memory - p)
                count = high_memory - p;
        written = 0;
 /* we don’t have page 0 mapped on x86/sparc.. */
#if defined(__i386__) || defined(__sparc__)
        while (PAGE_OFFSET + p < PAGE_SIZE && count > 0) {
                /* Hmm. Do something? */
                buf++;
                p++;
                count--;
                written++;
        }
#endif
        memcpy_fromfs((void *) p, buf, count);
        written += count;
        file->f_pos += written;
        return count;
}

static int mmap_mem(struct inode * inode, struct file * file,
                    struct vm_area_struct * vma)
{
        if (vma->vm_offset & ~PAGE_MASK)
                return -ENXIO;
#if defined(__i386__)
        /*
         * hmm.. This disables high-memory caching, as the XFree86 team
         * wondered about that at one time.
         * The surround logic should disable caching for the high device
         * addresses anyway, but right now this seems still needed.
         */
        if (x86 > 3 && vma->vm_offset >= high_memory)
```

```
                pgprot_val(vma->vm_page_prot) |= _PAGE_PCD;
#endif
        if (remap_page_range(vma->vm_start, vma->vm_offset,
                             vma->vm_end - vma->vm_start, vma->vm_page_prot))
                return -EAGAIN;
        vma->vm_inode = inode;
        inode->i_count++;
        return 0;
}

static int read_kmem(struct inode *inode, struct file *file, char *buf,
                     int count)
{
        int read1, read2;

        read1 = read_mem(inode, file, buf, count);
        if (read1 < 0)
                return read1;
        read2 = vread(buf + read1, (char *) ((unsigned long) file->f_pos),
                      count - read1);
        if (read2 < 0)
                return read2;
        file->f_pos += read2;
        return read1 + read2;
}

static int read_port(struct inode * inode, struct file * file,
                     char * buf, int count)
{
        unsigned int i = file->f_pos;
        char * tmp = buf;

        while (count-- > 0 && i < 65536) {
                put_user(inb(i),tmp);
                i++;
                tmp++;
        }
        file->f_pos = i;
        return tmp-buf;
}

static int write_port(struct inode * inode, struct file * file,
                      const char * buf, int count)
{
        unsigned int i = file->f_pos;
        const char * tmp = buf;

        while (count-- > 0 && i < 65536) {
                outb(get_user(tmp),i);
                i++;
                tmp++;
        }
        file->f_pos = i;
        return tmp-buf;
}

static int read_null(struct inode * node, struct file * file,
                     char * buf, int count)
{
        return 0;
```

```
}

static int write_null(struct inode * inode, struct file * file,
                      const char * buf, int count)
{
        return count;
}

static int read_zero(struct inode * node, struct file * file,
                     char * buf, int count)
{
        int left;

        for (left = count; left > 0; left--) {
                put_user(0,buf);
                buf++;
                if (need_resched)
                        schedule();
        }
        return count;
}

static int mmap_zero(struct inode * inode, struct file * file,
                     struct vm_area_struct * vma)
{
        if (vma->vm_flags & VM_SHARED)
                return -EINVAL;
        if (zeromap_page_range(vma->vm_start, vma->vm_end - vma->vm_start,
                               vma->vm_page_prot))
                return -EAGAIN;
        return 0;
}

static int read_full(struct inode * node, struct file * file,
                     char * buf, int count)
{
        file->f_pos += count;
        return count;
}

static int write_full(struct inode * inode, struct file * file,
                      const char * buf, int count)
{
        return -ENOSPC;
}

/*
 * Special lseek() function for /dev/null and /dev/zero.  Most
 * notably, you can fopen() both devices with "a" now.  This was
 * previously impossible.  SRB.
 */
static int null_lseek(struct inode * inode, struct file * file,
                      off_t offset, int orig)
{
        return file->f_pos=0;
}

/*
 * The memory devices use the full 32/64 bits of the offset, and so we
 * cannot check against negative addresses: they are ok. The return
```

```
 * value is weird, though, in that case (0).
 *
 * Also note that seeking relative to the "end of file" isn’t
 * supported: it has no meaning, so it returns -EINVAL.
 */
static int memory_lseek(struct inode * inode, struct file * file,
                        off_t offset, int orig)
{
        switch (orig) {
                case 0:
                        file->f_pos = offset;
                        return file->f_pos;
                case 1:
                        file->f_pos += offset;
                        return file->f_pos;
                default:
                        return -EINVAL;
        }
        if (file->f_pos < 0)
                return 0;
        return file->f_pos;
}

#define write_kmem      write_mem
#define mmap_kmem       mmap_mem
#define zero_lseek      null_lseek
#define write_zero      write_null

static struct file_operations ram_fops = {
        memory_lseek,
        read_ram,
        write_ram,
        NULL,           /* ram_readdir */
        NULL,           /* ram_select */
        NULL,           /* ram_ioctl */
        NULL,           /* ram_mmap */
        NULL,           /* no special open code */
        NULL,           /* no special release code */
        NULL            /* fsync */
};

static struct file_operations mem_fops = {
        memory_lseek,
        read_mem,
        write_mem,
        NULL,           /* mem_readdir */
        NULL,           /* mem_select */
        NULL,           /* mem_ioctl */
        mmap_mem,
        NULL,           /* no special open code */
        NULL,           /* no special release code */
        NULL            /* fsync */
};

static struct file_operations kmem_fops = {
        memory_lseek,
        read_kmem,
        write_kmem,
        NULL,           /* kmem_readdir */
        NULL,           /* kmem_select */
```

```
        NULL,           /* kmem_ioctl */
        mmap_kmem,
        NULL,           /* no special open code */
        NULL,           /* no special release code */
        NULL            /* fsync */
};

static struct file_operations null_fops = {
        null_lseek,
        read_null,
        write_null,
        NULL,           /* null_readdir */
        NULL,           /* null_select */
        NULL,           /* null_ioctl */
        NULL,           /* null_mmap */
        NULL,           /* no special open code */
        NULL,           /* no special release code */
        NULL            /* fsync */
};

static struct file_operations port_fops = {
        memory_lseek,
        read_port,
        write_port,
        NULL,           /* port_readdir */
        NULL,           /* port_select */
        NULL,           /* port_ioctl */
        NULL,           /* port_mmap */
        NULL,           /* no special open code */
        NULL,           /* no special release code */
        NULL            /* fsync */
};

static struct file_operations zero_fops = {
        zero_lseek,
        read_zero,
        write_zero,
        NULL,           /* zero_readdir */
        NULL,           /* zero_select */
        NULL,           /* zero_ioctl */
        mmap_zero,
        NULL,           /* no special open code */
        NULL            /* no special release code */
};

static struct file_operations full_fops = {
        memory_lseek,
        read_full,
        write_full,
        NULL,           /* full_readdir */
        NULL,           /* full_select */
        NULL,           /* full_ioctl */
        NULL,           /* full_mmap */
        NULL,           /* no special open code */
        NULL            /* no special release code */
};

static int memory_open(struct inode * inode, struct file * filp)
{
        switch (MINOR(inode->i_rdev)) {
```

```
		case 0:
			filp->f_op = &ram_fops;
			break;
		case 1:
			filp->f_op = &mem_fops;
			break;
		case 2:
			filp->f_op = &kmem_fops;
			break;
		case 3:
			filp->f_op = &null_fops;
			break;
		case 4:
			filp->f_op = &port_fops;
			break;
		case 5:
			filp->f_op = &zero_fops;
			break;
		case 7:
			filp->f_op = &full_fops;
			break;
		case 8:
			filp->f_op = &random_fops;
			break;
		case 9:
			filp->f_op = &urandom_fops;
			break;
		default:
			return -ENXIO;
	}
	if (filp->f_op && filp->f_op->open)
		return filp->f_op->open(inode,filp);
	return 0;
}

static struct file_operations memory_fops = {
	NULL,		/* lseek */
	NULL,		/* read */
	NULL,		/* write */
	NULL,		/* readdir */
	NULL,		/* select */
	NULL,		/* ioctl */
	NULL,		/* mmap */
	memory_open,	/* just a selector for the real open */
	NULL,		/* release */
	NULL		/* fsync */
};

int chr_dev_init(void)
{
	if (register_chrdev(MEM_MAJOR,"mem",&memory_fops))
		printk("unable to get major %d for memory devs\n",
		       MEM_MAJOR);
	rand_initialize();
	tty_init();
#ifdef CONFIG_PRINTER
	lp_init();
#endif
#if defined (CONFIG_BUSMOUSE) || defined(CONFIG_UMISC) || \
    defined (CONFIG_PSMOUSE) || defined (CONFIG_MS_BUSMOUSE) || \
```

```
    defined (CONFIG_ATIXL_BUSMOUSE) || defined(CONFIG_SOFT_WATCHDOG) || \
    defined (CONFIG_PCWATCHDOG) || \
    defined (CONFIG_APM) || defined (CONFIG_RTC) || \
    defined (CONFIG_SUN_MOUSE)
        misc_init();
#endif
#ifdef CONFIG_SOUND
        soundcard_init();
#endif
#if CONFIG_QIC02_TAPE
        qic02_tape_init();
#endif
#if CONFIG_ISDN
        isdn_init();
#endif
#ifdef CONFIG_FTAPE
        ftape_init();
#endif
        return 0;
}
```