```
/*
 *  linux/kernel/blk_drv/ramdisk.c
 *
 *  Written by Theodore Ts'o, 12/2/91
 *
 * Modifications by Fred N. van Kempen to allow for bootable root
 * disks (which are used in LINUX/Pro).  Also some cleanups.  03/03/93
 */


#include <linux/sched.h>
#include <linux/minix_fs.h>
#include <linux/ext2_fs.h>
#include <linux/fs.h>
#include <linux/kernel.h>
#include <linux/string.h>
#include <linux/mm.h>

#include <asm/system.h>
#include <asm/segment.h>

#define MAJOR_NR  MEM_MAJOR
#include "blk.h"

#define RAMDISK_MINOR   1

extern void wait_for_keypress(void);

char    *rd_start;
int     rd_length = 0;
static int rd_blocksizes[2] = {0, 0};

static void do_rd_request(void)
{
        int     len;
        char    *addr;

repeat:
        INIT_REQUEST;
        addr = rd_start + (CURRENT->sector << 9);
        len = CURRENT->current_nr_sectors << 9;

        if ((MINOR(CURRENT->dev) != RAMDISK_MINOR) ||
            (addr+len > rd_start+rd_length)) {
                end_request(0);
                goto repeat;
        }
        if (CURRENT-> cmd == WRITE) {
                (void ) memcpy(addr,
                              CURRENT->buffer,
                              len);
        } else if (CURRENT->cmd == READ) {
                (void) memcpy(CURRENT->buffer,
                              addr,
                              len);
        } else
                panic("RAMDISK: unknown RAM disk command !\n");
        end_request(1);
        goto repeat;
}
```

```
static struct file_operations rd_fops = {
        NULL,                   /* lseek - default */
        block_read,             /* read - general block-dev read */
        block_write,            /* write - general block-dev write */
        NULL,                   /* readdir - bad */
        NULL,                   /* select */
        NULL,                   /* ioctl */
        NULL,                   /* mmap */
        NULL,                   /* no special open code */
        NULL,                   /* no special release code */
        block_fsync             /* fsync */
};

/*
 * Returns amount of memory which needs to be reserved.
 */
long rd_init(long mem_start, int length)
{
        int     i;
        char    *cp;

        if (register_blkdev(MEM_MAJOR,"rd",&rd_fops)) {
                printk("RAMDISK: Unable to get major %d.\n", MEM_MAJOR);
                return 0;
        }
        blk_dev[MEM_MAJOR].request_fn = DEVICE_REQUEST;
        rd_start = (char *) mem_start;
        rd_length = length;
        cp = rd_start;
        for (i=0; i < length; i++)
                *cp++ = '\0';

        for(i=0;i<2;i++) rd_blocksizes[i] = 1024;
        blksize_size[MAJOR_NR] = rd_blocksizes;

        return(length);
}

static void do_load(void)
{
        struct buffer_head *bh;
        struct super_block {
          union
          {
            char minix [sizeof (struct minix_super_block)];
            char ext2 [sizeof (struct ext2_super_block)];
          } record;
        } sb;
        struct minix_super_block *minixsb =
                (struct minix_super_block *)&sb;
        struct ext2_super_block *ext2sb =
                (struct ext2_super_block *)&sb;
        int             block, tries;
        int             i = 1;
        int             nblocks;
        char            *cp;

        /*
         * Check for a super block on the diskette.
```

```
	 * The old-style boot/root diskettes had their RAM image
	 * starting at block 512 of the boot diskette.  LINUX/Pro
	 * uses the entire diskette as a file system, so in that
	 * case, we have to look at block 0.  Be intelligent about
	 * this, and check both... - FvK
	 */
	for (tries = 0; tries < 1000; tries += 512) {
		block = tries;
		bh = breada(ROOT_DEV,block+1,BLOCK_SIZE, 0,  PAGE_SIZE);
		if (!bh) {
			printk("RAMDISK: I/O error while looking "
			       "for super block!\n");
			return;
		}

		/*
		 * This is silly- why do we require it to be a
		 * MINIX FS?
		 */
		*((struct super_block *) &sb) =
			*((struct super_block *) bh->b_data);
		brelse(bh);


		/* Try Minix */
		nblocks = -1;
		if (minixsb->s_magic == MINIX_SUPER_MAGIC ||
		    minixsb->s_magic == MINIX_SUPER_MAGIC2) {
			printk("RAMDISK: Minix filesystem found "
			       "at block %d\n", block);
			nblocks = (minixsb->s_nzones <<
				   minixsb->s_log_zone_size);
		}

		/* Try ext2 */
		if (nblocks == -1 && (ext2sb->s_magic ==
			EXT2_PRE_02B_MAGIC ||
			ext2sb->s_magic == EXT2_SUPER_MAGIC))
		{
			printk("RAMDISK: Ext2 filesystem found "
			       "at block %d\n", block);
			nblocks = ext2sb->s_blocks_count;
		}

		if (nblocks == -1)
		{
			printk("RAMDISK: trying old-style RAM image.\n");
			continue;
		}

		if (nblocks > (rd_length >> BLOCK_SIZE_BITS)) {
			printk("RAMDISK: image too big! "
			       "(%d/%d blocks)\n", nblocks,
			       rd_length >> BLOCK_SIZE_BITS);
			return;
		}
		printk("RAMDISK: Loading %d blocks into RAM disk",
		       nblocks);

		/* We found an image file system.  Load it into core! */
```

```
                cp = rd_start;
                while (nblocks) {
                        if (nblocks > 2)
                                bh = breada(ROOT_DEV, block, BLOCK_SIZE,
                                            0, PAGE_SIZE);
                        else
                                bh = bread(ROOT_DEV, block, BLOCK_SIZE);
                        if (!bh) {
                                printk("RAMDISK: I/O error on block "
                                       "%d, aborting!\n", block);
                                return;
                        }
                        (void) memcpy(cp, bh->b_data, BLOCK_SIZE);
                        brelse(bh);
                        if (!(nblocks-- & 15)) printk(".");
                        cp += BLOCK_SIZE;
                        block++;
                        i++;
                }
                printk("\ndone\n");

                /*
                 * We loaded the file system image.  Prepare for
                 * mounting it.
                 */
                ROOT_DEV = ((MEM_MAJOR << 8) | RAMDISK_MINOR);
                return;
        }
}

/*
 * If the root device is the RAM disk, try to load it.
 * In order to do this, the root device is originally set to the
 * floppy, and we later change it to be RAM disk.
 */
void rd_load(void)
{
        struct inode inode;
        struct file filp;

        /* If no RAM disk specified, give up early. */
        if (!rd_length)
                return;
        printk("RAMDISK: %d bytes, starting at 0x%p\n",
                        rd_length, rd_start);

        /*
         * If we are doing a diskette boot, we might have to
         * pre-load it.
         */
        if (MAJOR(ROOT_DEV) != FLOPPY_MAJOR)
                return;

        /* for Slackware install disks */
        printk(KERN_NOTICE "VFS: Insert ramdisk floppy "
               "and press ENTER\n");
        wait_for_keypress();

        memset(&filp, 0, sizeof(filp));
        memset(&inode, 0, sizeof(inode));
```

```
        inode.i_rdev = ROOT_DEV;
        filp.f_mode = 1; /* read only */
        filp.f_inode = &inode;
        if(blkdev_open(&inode, &filp) == 0 ){
                do_load();
                if(filp.f_op && filp.f_op->release)
                        filp.f_op->release(&inode,&filp);
        }
}
```