```
/*
 * (C) Copyright 2000-2002
 * Wolfgang Denk, DENX Software Engineering, wd@denx.de.
 *
 * See file CREDITS for list of people who contributed to this
 * project.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License as
 * published by the Free Software Foundation; either version 2 of
 * the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston,
 * MA 02111-1307 USA
 */

/*
 * Boot support
 */
#include <common.h>
#include <watchdog.h>
#include <command.h>
#include <cmd_boot.h>
#include <image.h>
#include <malloc.h>
#include <zlib.h>
#include <environment.h>
#include <asm/byteorder.h>
#if (CONFIG_COMMANDS & CFG_CMD_DATE) || defined(CONFIG_TIMESTAMP)
#include <rtc.h>
#endif

#ifdef CFG_HUSH_PARSER
#include <hush.h>
#endif

#ifdef CONFIG_SHOW_BOOT_PROGRESS
# include <status_led.h>
# define SHOW_BOOT_PROGRESS(arg)        show_boot_progress(arg)
#else
# define SHOW_BOOT_PROGRESS(arg)
#endif

#ifdef CFG_INIT_RAM_LOCK
#include <asm/cache.h>
#endif

#ifdef CONFIG_LOGBUFFER
#include <logbuff.h>
#endif

#ifdef CONFIG_HAS_DATAFLASH
#include <dataflash.h>
#endif

/*
 * Some systems (for example LWMON) have very short watchdog periods;
 * we must make sure to split long operations like memmove() or
 * crc32() into reasonable chunks.
 */
#if defined(CONFIG_HW_WATCHDOG) || defined(CONFIG_WATCHDOG)
# define CHUNKSZ (64 * 1024)
#endif

int  gunzip (void *, int, unsigned char *, int *);

static void *zalloc(void *, unsigned, unsigned);
static void zfree(void *, void *, unsigned);
```

```
#if (CONFIG_COMMANDS & CFG_CMD_IMI)
static int image_info (unsigned long addr);
#endif
static void print_type (image_header_t *hdr);

#ifdef __I386__
image_header_t *fake_header(image_header_t *hdr, void *ptr, int size);
#endif

/*
 *  Continue booting an OS image; caller already has:
 *  - copied image header to global variable `header'
 *  - checked header magic number, checksums (both header & image),
 *  - verified image architecture (PPC) and type (KERNEL or MULTI),
 *  - loaded (first part of) image to header load address,
 *  - disabled interrupts.
 */
typedef void boot_os_Fcn (cmd_tbl_t *cmdtp, int flag,
                          int   argc, char *argv[],
                          ulong addr,           /* of image to boot */
                          ulong *len_ptr,       /* multi-file image length table */
                          int   verify);        /* getenv("verify")[0] != 'n' */

/* ghcstop */
#ifdef CONFIG_PPC
static boot_os_Fcn do_bootm_linux;
#else
extern boot_os_Fcn do_bootm_linux; /* arm용 부트함수는 lib_arm/armlinux.c에 있음 */
#endif
static boot_os_Fcn do_bootm_netbsd;
static boot_os_Fcn do_bootm_rtems;
#if (CONFIG_COMMANDS & CFG_CMD_ELF)
static boot_os_Fcn do_bootm_vxworks;
static boot_os_Fcn do_bootm_qnxelf;
int do_bootvx ( cmd_tbl_t *cmdtp, int flag, int argc, char *argv[] );
int do_bootelf (cmd_tbl_t *cmdtp, int flag, int argc, char *argv[] );
#endif /* CFG_CMD_ELF */
#if defined(CONFIG_ARTOS) && defined(CONFIG_PPC)
static boot_os_Fcn do_bootm_artos;
#endif

image_header_t header;

/* ghcstop */
/* bootm을 아무 옵션없이 명령을 내렸을 경우 이 번지에 커널 이미지가 로딩되어 있다고
보고 여기다가 로드를 해버린다. 만일 옵션을 준다면 그 번지에 커널이미지가 로딩되어
있다고 보고 해당번지에서 커널 이미지를 찾아서 이미지 헤더 파악하고 crc체킹하고
압축을 mkimage시 -a 로 지정된 번지로 풀고, -e 로 지정된 번지에서부터 커널을
시작하게 된다*/
ulong load_addr = CFG_LOAD_ADDR;               /* Default Load Address */

/* ghcstop */
/* u-boot/tools 디렉토리에 있는 mkimage로 만들어진 커널을 읽어서 헤더를 보고
어떤 함수로 부팅을 시켜야 하는지(어느 os로) 결정을 한다...*/
int do_bootm (cmd_tbl_t *cmdtp, int flag, int argc, char *argv[])
{
        ulong   iflag;
        ulong   addr;
        ulong   data, len, checksum;
        ulong  *len_ptr;
        int     i, verify;
        char    *name, *s;
        int     (*appl)(cmd_tbl_t *, int, int, char *[]);
        image_header_t *hdr = &header;

        s = getenv ("verify");
        verify = (s && (*s == 'n')) ? 0 : 1;

        if (argc < 2) {
                addr = load_addr;
        } else {
                addr = simple_strtoul(argv[1], NULL, 16);
        }

        SHOW_BOOT_PROGRESS (1);
        printf ("## Booting image at %08lx ...\n", addr);
```

```
        /* Copy header so we can blank CRC field for re-calculation */
#ifdef CONFIG_HAS_DATAFLASH
        if (addr_dataflash(addr)){
                read_dataflash(addr, sizeof(image_header_t), (char *)&header);
        } else
#endif
        memmove (&header, (char *)addr, sizeof(image_header_t));

        if (ntohl(hdr->ih_magic) != IH_MAGIC) {
#ifdef __I386__ /* correct image format not implemented yet - fake it */
                if (fake_header(hdr, (void*)addr, -1) != NULL) {
                        /* to compensate for the addition below */
                        addr -= sizeof(image_header_t);
                        /* turnof verify,
                         * fake_header() does not fake the data crc
                         */
                        verify = 0;
                } else
#endif  /* __I386__ */
                {
                    printf ("Bad Magic Number\n");
                    SHOW_BOOT_PROGRESS (-1);
                    return 1;
                }
        }
        SHOW_BOOT_PROGRESS (2);

        data = (ulong)&header;
        len  = sizeof(image_header_t);

        checksum = ntohl(hdr->ih_hcrc);
        hdr->ih_hcrc = 0;

        if (crc32 (0, (char *)data, len) != checksum) {
                printf ("Bad Header Checksum\n");
                SHOW_BOOT_PROGRESS (-2);
                return 1;
        }
        SHOW_BOOT_PROGRESS (3);

        /* for multi-file images we need the data part, too */
        print_image_hdr (hdr);

        /* 실제 압축된 커널이 위치한데로 포인터를 옮긴다 */
        data = addr + sizeof(image_header_t);
        len  = ntohl(hdr->ih_size);

#ifdef CONFIG_HAS_DATAFLASH
        if (addr_dataflash(addr)){
                read_dataflash(data, len, (char *)CFG_LOAD_ADDR);
                data = CFG_LOAD_ADDR;
        }
#endif

        if (verify) {
                printf ("   Verifying Checksum ... ");
                if (crc32 (0, (char *)data, len) != ntohl(hdr->ih_dcrc)) {
                        printf ("Bad Data CRC\n");
                        SHOW_BOOT_PROGRESS (-3);
                        return 1;
                }
                printf ("OK\n");
        }
        SHOW_BOOT_PROGRESS (4);

        len_ptr = (ulong *)data;

#if defined(__PPC__)
        if (hdr->ih_arch != IH_CPU_PPC)
#elif defined(__ARM__)
        if (hdr->ih_arch != IH_CPU_ARM)
#elif defined(__I386__)
        if (hdr->ih_arch != IH_CPU_I386)
#elif defined(__mips__)
        if (hdr->ih_arch != IH_CPU_MIPS)
```

```
#else
# error Unknown CPU type
#endif
        {
                printf ("Unsupported Architecture 0x%x\n", hdr->ih_arch);
                SHOW_BOOT_PROGRESS (-4);
                return 1;
        }
        SHOW_BOOT_PROGRESS (5);

        switch (hdr->ih_type) {
        case IH_TYPE_STANDALONE:        name = "Standalone Application";
                                        break;
        case IH_TYPE_KERNEL:            name = "Kernel Image";
                                        break;
        case IH_TYPE_MULTI:             name = "Multi-File Image";
                                        len  = ntohl(len_ptr[0]);
                                        /* OS kernel is always the first image */
                                        data += 8; /* kernel_len + terminator */
                                        for (i=1; len_ptr[i]; ++i)
                                                data += 4;
                                        break;
        default: printf ("Wrong Image Type for %s command\n", cmdtp->name);
                SHOW_BOOT_PROGRESS (-5);
                return 1;
        }
        SHOW_BOOT_PROGRESS (6);

        /*
         * We have reached the point of no return: we are going to
         * overwrite all exception vector code, so we cannot easily
         * recover from any failures any more...
         */

        iflag = disable_interrupts();

#ifdef CONFIG_AMIGAONEG3SE
        /*
         * We've possible left the caches enabled during
         * bios emulation, so turn them off again
         */
        icache_disable();
        invalidate_l1_instruction_cache();
        flush_data_cache();
        dcache_disable();
#endif

        switch (hdr->ih_comp) {
        case IH_COMP_NONE:
                if(ntohl(hdr->ih_load) == addr) {
                        printf ("   XIP %s ... ", name);
                } else {
#if defined(CONFIG_HW_WATCHDOG) || defined(CONFIG_WATCHDOG)
                        size_t l = len;
                        void *to = (void *)ntohl(hdr->ih_load);
                        void *from = (void *)data;

                        printf ("   Loading %s ... ", name);

                        while (l > 0) {
                                size_t tail = (l > CHUNKSZ) ? CHUNKSZ : l;
                                WATCHDOG_RESET();
                                memmove (to, from, tail);
                                to += tail;
                                from += tail;
                                l -= tail;
                        }
#else   /* !(CONFIG_HW_WATCHDOG || CONFIG_WATCHDOG) */
                        memmove ((void *) ntohl(hdr->ih_load), (uchar *)data, len);
#endif  /* CONFIG_HW_WATCHDOG || CONFIG_WATCHDOG */
                }
                break;
        case IH_COMP_GZIP:
                /* ghcstop *** 중요함 ***
                 * 압축을 풀어버린다...중요한 부분 arm 리눅스 커널에서도
                 * 압축을 여기서 풀어버린다....원래 arm용 커널들은 주로 압축을
```

```
                 * 커널 자체의 부트로더에서 하는데....
                 * u-boot는 압축을 여기서 풀어버리기 때문에 그 딴짓을 할 필요가 없다
                 * www.denx.de의 s3c2400용 커널인 linuxarm쪽의 arch/arm/밑의
                 * Makefile들을 살펴 볼 것
                 */
                printf ("   Uncompressing %s ... ", name);

                /* 원래 커널의 이미지가 로드되어야 할 주소로 압축을 풀어서 옮긴다 */
                if (gunzip ((void *)ntohl(hdr->ih_load), 0x400000,
                            (uchar *)data, (int *)&len) != 0) {
                        printf ("GUNZIP ERROR - must RESET board to recover\n");
                        SHOW_BOOT_PROGRESS (-6);
                        do_reset (cmdtp, flag, argc, argv);
                }
                break;
        default:
                if (iflag)
                        enable_interrupts();
                printf ("Unimplemented compression type %d\n", hdr->ih_comp);
                SHOW_BOOT_PROGRESS (-7);
                return 1;
        }
        printf ("OK\n");
        SHOW_BOOT_PROGRESS (7);

        switch (hdr->ih_type) {
        case IH_TYPE_STANDALONE:
                if (iflag)
                        enable_interrupts();

                /* load (and uncompress), but don't start if "autostart"
                 * is set to "no"
                 */
                if (((s = getenv("autostart")) != NULL) && (strcmp(s,"no") == 0))
                        return 0;
                appl = (int (*)(cmd_tbl_t *, int, int, char *[]))ntohl(hdr->ih_ep);
                (*appl)(cmdtp, flag, argc-1, &argv[1]);
                return 0;
        case IH_TYPE_KERNEL:
        case IH_TYPE_MULTI:
                /* handled below */
                break;
        default:
                if (iflag)
                        enable_interrupts();
                printf ("Can't boot image type %d\n", hdr->ih_type);
                SHOW_BOOT_PROGRESS (-8);
                return 1;
        }
        SHOW_BOOT_PROGRESS (8);

        /* ghcstop: 지정된 os에 대한 부트루틴을 시작한다 */
        switch (hdr->ih_os) {
        default:                        /* handled by (original) Linux case */
        case IH_OS_LINUX:
            do_bootm_linux  (cmdtp, flag, argc, argv,
                             addr, len_ptr, verify); /* lib_arm/armlinux.c의 함수호출 */
            break;
        case IH_OS_NETBSD:
            do_bootm_netbsd (cmdtp, flag, argc, argv,
                             addr, len_ptr, verify);
            break;

        case IH_OS_RTEMS:
            do_bootm_rtems (cmdtp, flag, argc, argv,
                             addr, len_ptr, verify);
            break;

#if (CONFIG_COMMANDS & CFG_CMD_ELF)
        case IH_OS_VXWORKS:
            do_bootm_vxworks (cmdtp, flag, argc, argv,
                              addr, len_ptr, verify);
            break;
        case IH_OS_QNX:
            do_bootm_qnxelf (cmdtp, flag, argc, argv,
                              addr, len_ptr, verify);
```

```
            break;
#endif /* CFG_CMD_ELF */
#ifdef CONFIG_ARTOS
        case IH_OS_ARTOS:
            do_bootm_artos  (cmdtp, flag, argc, argv,
                             addr, len_ptr, verify);
            break;
#endif
        }

        SHOW_BOOT_PROGRESS (-9);
#ifdef DEBUG
        printf ("\n## Control returned to monitor - resetting...\n");
        do_reset (cmdtp, flag, argc, argv);
#endif
        return 1;
}

#ifdef CONFIG_PPC
static void
do_bootm_linux (cmd_tbl_t *cmdtp, int flag,
                int     argc, char *argv[],
                ulong   addr,
                ulong   *len_ptr,
                int     verify)
{
        DECLARE_GLOBAL_DATA_PTR;

        ulong   sp;
        ulong   len, checksum;
        ulong   initrd_start, initrd_end;
        ulong   cmd_start, cmd_end;
        ulong   initrd_high;
        ulong   data;
        int     initrd_copy_to_ram = 1;
        char    *cmdline;
        char    *s;
        bd_t    *kbd;
        void    (*kernel)(bd_t *, ulong, ulong, ulong, ulong);
        image_header_t *hdr = &header;

        if ((s = getenv ("initrd_high")) != NULL) {
                /* a value of "no" or a similar string will act like 0,
                 * turning the "load high" feature off. This is intentional.
                 */
                initrd_high = simple_strtoul(s, NULL, 16);
                if (initrd_high == ~0)
                        initrd_copy_to_ram = 0;
        } else {        /* not set, no restrictions to load high */
                initrd_high = ~0;
        }

#ifdef CONFIG_LOGBUFFER
        kbd=gd->bd;
        /* Prevent initrd from overwriting logbuffer */
        if (initrd_high < (kbd->bi_memsize-LOGBUFF_LEN-LOGBUFF_OVERHEAD))
                initrd_high = kbd->bi_memsize-LOGBUFF_LEN-LOGBUFF_OVERHEAD;
        debug ("## Logbuffer at 0x%08lX ", kbd->bi_memsize-LOGBUFF_LEN);
#endif

        /*
         * Booting a (Linux) kernel image
         *
         * Allocate space for command line and board info - the
         * address should be as high as possible within the reach of
         * the kernel (see CFG_BOOTMAPSZ settings), but in unused
         * memory, which means far enough below the current stack
         * pointer.
         */

        asm( "mr %0,1": "=r"(sp) : );

        debug ("## Current stack ends at 0x%08lX ", sp);

        sp -= 2048;             /* just to be sure */
        if (sp > CFG_BOOTMAPSZ)
```

```
                sp = CFG_BOOTMAPSZ;
        sp &= ~0xF;

        debug ("=> set upper limit to 0x%08lX\n", sp);

        cmdline = (char *)((sp - CFG_BARGSIZE) & ~0xF);
        kbd = (bd_t *)(((ulong)cmdline - sizeof(bd_t)) & ~0xF);

        if ((s = getenv("bootargs")) == NULL)
                s = "";

        strcpy (cmdline, s);

        cmd_start    = (ulong)&cmdline[0];
        cmd_end      = cmd_start + strlen(cmdline);

        *kbd = *(gd->bd);

#ifdef  DEBUG
        printf ("## cmdline at 0x%08lX ... 0x%08lX\n", cmd_start, cmd_end);

        do_bdinfo (NULL, 0, 0, NULL);
#endif

        if ((s = getenv ("clocks_in_mhz")) != NULL) {
                /* convert all clock information to MHz */
                kbd->bi_intfreq /= 1000000L;
                kbd->bi_busfreq /= 1000000L;
#if defined(CONFIG_8260)
                kbd->bi_cpmfreq /= 1000000L;
                kbd->bi_brgfreq /= 1000000L;
                kbd->bi_sccfreq /= 1000000L;
                kbd->bi_vco     /= 1000000L;
#endif /* CONFIG_8260 */
        }

        kernel = (void (*)(bd_t *, ulong, ulong, ulong, ulong))hdr->ih_ep;

        /*
         * Check if there is an initrd image
         */
        if (argc >= 3) {
                SHOW_BOOT_PROGRESS (9);

                addr = simple_strtoul(argv[2], NULL, 16);

                printf ("## Loading RAMDisk Image at %08lx ...\n", addr);

                /* Copy header so we can blank CRC field for re-calculation */
                memmove (&header, (char *)addr, sizeof(image_header_t));

                if (hdr->ih_magic  != IH_MAGIC) {
                        printf ("Bad Magic Number\n");
                        SHOW_BOOT_PROGRESS (-10);
                        do_reset (cmdtp, flag, argc, argv);
                }

                data = (ulong)&header;
                len  = sizeof(image_header_t);

                checksum = hdr->ih_hcrc;
                hdr->ih_hcrc = 0;

                if (crc32 (0, (char *)data, len) != checksum) {
                        printf ("Bad Header Checksum\n");
                        SHOW_BOOT_PROGRESS (-11);
                        do_reset (cmdtp, flag, argc, argv);
                }

                SHOW_BOOT_PROGRESS (10);

                print_image_hdr (hdr);

                data = addr + sizeof(image_header_t);
                len  = hdr->ih_size;
```

```
                if (verify) {
                        ulong csum = 0;
#if defined(CONFIG_HW_WATCHDOG) || defined(CONFIG_WATCHDOG)
                        ulong cdata = data, edata = cdata + len;
#endif  /* CONFIG_HW_WATCHDOG || CONFIG_WATCHDOG */

                        printf ("   Verifying Checksum ... ");

#if defined(CONFIG_HW_WATCHDOG) || defined(CONFIG_WATCHDOG)

                        while (cdata < edata) {
                                ulong chunk = edata - cdata;

                                if (chunk > CHUNKSZ)
                                        chunk = CHUNKSZ;
                                csum = crc32 (csum, (char *)cdata, chunk);
                                cdata += chunk;

                                WATCHDOG_RESET();
                        }
#else   /* !(CONFIG_HW_WATCHDOG || CONFIG_WATCHDOG) */
                        csum = crc32 (0, (char *)data, len);
#endif  /* CONFIG_HW_WATCHDOG || CONFIG_WATCHDOG */

                        if (csum != hdr->ih_dcrc) {
                                printf ("Bad Data CRC\n");
                                SHOW_BOOT_PROGRESS (-12);
                                do_reset (cmdtp, flag, argc, argv);
                        }
                        printf ("OK\n");
                }

                SHOW_BOOT_PROGRESS (11);

                if ((hdr->ih_os   != IH_OS_LINUX)       ||
                    (hdr->ih_arch != IH_CPU_PPC)        ||
                    (hdr->ih_type != IH_TYPE_RAMDISK)   ) {
                        printf ("No Linux PPC Ramdisk Image\n");
                        SHOW_BOOT_PROGRESS (-13);
                        do_reset (cmdtp, flag, argc, argv);
                }

                /*
                 * Now check if we have a multifile image
                 */
        } else if ((hdr->ih_type==IH_TYPE_MULTI) && (len_ptr[1])) {
                u_long tail    = ntohl(len_ptr[0]) % 4;
                int i;

                SHOW_BOOT_PROGRESS (13);

                /* skip kernel length and terminator */
                data = (ulong)(&len_ptr[2]);
                /* skip any additional image length fields */
                for (i=1; len_ptr[i]; ++i)
                        data += 4;
                /* add kernel length, and align */
                data += ntohl(len_ptr[0]);
                if (tail) {
                        data += 4 - tail;
                }

                len   = ntohl(len_ptr[1]);

        } else {
                /*
                 * no initrd image
                 */
                SHOW_BOOT_PROGRESS (14);

                len = data = 0;
        }

        if (!data) {
                debug ("No initrd\n");
        }
```

```
        if (data) {
            if (!initrd_copy_to_ram) {  /* zero-copy ramdisk support */
                initrd_start = data;
                initrd_end = initrd_start + len;
            } else {
                initrd_start  = (ulong)kbd - len;
                initrd_start &= ~(4096 - 1);    /* align on page */

                if (initrd_high) {
                        ulong nsp;

                        /*
                         * the inital ramdisk does not need to be within
                         * CFG_BOOTMAPSZ as it is not accessed until after
                         * the mm system is initialised.
                         *
                         * do the stack bottom calculation again and see if
                         * the initrd will fit just below the monitor stack
                         * bottom without overwriting the area allocated
                         * above for command line args and board info.
                         */
                        asm( "mr %0,1": "=r"(nsp) : );
                        nsp -= 2048;            /* just to be sure */
                        nsp &= ~0xF;
                        if (nsp > initrd_high)  /* limit as specified */
                                nsp = initrd_high;
                        nsp -= len;
                        nsp &= ~(4096 - 1);     /* align on page */
                        if (nsp >= sp)
                                initrd_start = nsp;
                }

                SHOW_BOOT_PROGRESS (12);

                debug ("## initrd at 0x%08lX ... 0x%08lX (len=%ld=0x%lX)\n",
                        data, data + len - 1, len, len);

                initrd_end    = initrd_start + len;
                printf ("   Loading Ramdisk to %08lx, end %08lx ... ",
                        initrd_start, initrd_end);
#if defined(CONFIG_HW_WATCHDOG) || defined(CONFIG_WATCHDOG)
                {
                        size_t l = len;
                        void *to = (void *)initrd_start;
                        void *from = (void *)data;

                        while (l > 0) {
                                size_t tail = (l > CHUNKSZ) ? CHUNKSZ : l;
                                WATCHDOG_RESET();
                                memmove (to, from, tail);
                                to += tail;
                                from += tail;
                                l -= tail;
                        }
                }
#else   /* !(CONFIG_HW_WATCHDOG || CONFIG_WATCHDOG) */
                memmove ((void *)initrd_start, (void *)data, len);
#endif  /* CONFIG_HW_WATCHDOG || CONFIG_WATCHDOG */
                printf ("OK\n");
            }
        } else {
                initrd_start = 0;
                initrd_end = 0;
        }


        debug ("## Transferring control to Linux (at address %08lx) ...\n",
                (ulong)kernel);

        SHOW_BOOT_PROGRESS (15);

#ifdef CFG_INIT_RAM_LOCK
        unlock_ram_in_cache();
#endif
        /*
```

```
         * Linux Kernel Parameters:
         *   r3: ptr to board info data
         *   r4: initrd_start or 0 if no initrd
         *   r5: initrd_end - unused if r4 is 0
         *   r6: Start of command line string
         *   r7: End   of command line string
         */
        (*kernel) (kbd, initrd_start, initrd_end, cmd_start, cmd_end);
}
#endif /* CONFIG_PPC */

static void
do_bootm_netbsd (cmd_tbl_t *cmdtp, int flag,
                 int     argc, char *argv[],
                 ulong   addr,
                 ulong   *len_ptr,
                 int     verify)
{
        DECLARE_GLOBAL_DATA_PTR;

        image_header_t *hdr = &header;

        void    (*loader)(bd_t *, image_header_t *, char *, char *);
        image_header_t *img_addr;
        char     *consdev;
        char     *cmdline;


        /*
         * Booting a (NetBSD) kernel image
         *
         * This process is pretty similar to a standalone application:
         * The (first part of an multi-) image must be a stage-2 loader,
         * which in turn is responsible for loading & invoking the actual
         * kernel.  The only differences are the parameters being passed:
         * besides the board info strucure, the loader expects a command
         * line, the name of the console device, and (optionally) the
         * address of the original image header.
         */

        img_addr = 0;
        if ((hdr->ih_type==IH_TYPE_MULTI) && (len_ptr[1]))
                img_addr = (image_header_t *) addr;


        consdev = "";
#if   defined (CONFIG_8xx_CONS_SMC1)
        consdev = "smc1";
#elif defined (CONFIG_8xx_CONS_SMC2)
        consdev = "smc2";
#elif defined (CONFIG_8xx_CONS_SCC2)
        consdev = "scc2";
#elif defined (CONFIG_8xx_CONS_SCC3)
        consdev = "scc3";
#endif

        if (argc > 2) {
                ulong len;
                int   i;

                for (i=2, len=0 ; i<argc ; i+=1)
                        len += strlen (argv[i]) + 1;
                cmdline = malloc (len);

                for (i=2, len=0 ; i<argc ; i+=1) {
                        if (i > 2)
                                cmdline[len++] = ' ';
                        strcpy (&cmdline[len], argv[i]);
                        len += strlen (argv[i]);
                }
        } else if ((cmdline = getenv("bootargs")) == NULL) {
                cmdline = "";
        }

        loader = (void (*)(bd_t *, image_header_t *, char *, char *)) hdr->ih_ep;
```

```
        printf ("## Transferring control to NetBSD stage-2 loader (at address %08lx) ...\n",
                (ulong)loader);

        SHOW_BOOT_PROGRESS (15);

        /*
         * NetBSD Stage-2 Loader Parameters:
         *   r3: ptr to board info data
         *   r4: image address
         *   r5: console device
         *   r6: boot args string
         */
        (*loader) (gd->bd, img_addr, consdev, cmdline);
}

#if defined(CONFIG_ARTOS) && defined(CONFIG_PPC)

/* Function that returns a character from the environment */
extern uchar (*env_get_char)(int);

static void
do_bootm_artos (cmd_tbl_t *cmdtp, int flag,
                int     argc, char *argv[],
                ulong   addr,
                ulong   *len_ptr,
                int     verify)
{
        DECLARE_GLOBAL_DATA_PTR;
        ulong top;
        char *s, *cmdline;
        char **fwenv, **ss;
        int i, j, nxt, len, envno, envsz;
        bd_t *kbd;
        void (*entry)(bd_t *bd, char *cmdline, char **fwenv, ulong top);
        image_header_t *hdr = &header;

        /*
         * Booting an ARTOS kernel image + application
         */

        /* this used to be the top of memory, but was wrong... */
#ifdef CONFIG_PPC
        /* get stack pointer */
        asm volatile ("mr %0,1" : "=r"(top) );
#endif
        debug ("## Current stack ends at 0x%08lX ", top);

        top -= 2048;            /* just to be sure */
        if (top > CFG_BOOTMAPSZ)
                top = CFG_BOOTMAPSZ;
        top &= ~0xF;

        debug ("=> set upper limit to 0x%08lX\n", top);

        /* first check the artos specific boot args, then the linux args*/
        if ((s = getenv("abootargs")) == NULL && (s = getenv("bootargs")) == NULL)
                s = "";

        /* get length of cmdline, and place it */
        len = strlen(s);
        top = (top - (len + 1)) & ~0xF;
        cmdline = (char *)top;
        debug ("## cmdline at 0x%08lX ", top);
        strcpy(cmdline, s);

        /* copy bdinfo */
        top = (top - sizeof(bd_t)) & ~0xF;
        debug ("## bd at 0x%08lX ", top);
        kbd = (bd_t *)top;
        memcpy(kbd, gd->bd, sizeof(bd_t));

        /* first find number of env entries, and their size */
        envno = 0;
        envsz = 0;
        for (i = 0; env_get_char(i) != '\0'; i = nxt + 1) {
                for (nxt = i; env_get_char(nxt) != '\0'; ++nxt)
```

```
                        ;
                envno++;
                envsz += (nxt - i) + 1; /* plus trailing zero */
        }
        envno++;        /* plus the terminating zero */
        debug ("## %u envvars total size %u ", envno, envsz);

        top = (top - sizeof(char **)*envno) & ~0xF;
        fwenv = (char **)top;
        debug ("## fwenv at 0x%08lX ", top);

        top = (top - envsz) & ~0xF;
        s = (char *)top;
        ss = fwenv;

        /* now copy them */
        for (i = 0; env_get_char(i) != '\0'; i = nxt + 1) {
                for (nxt = i; env_get_char(nxt) != '\0'; ++nxt)
                        ;
                *ss++ = s;
                for (j = i; j < nxt; ++j)
                        *s++ = env_get_char(j);
                *s++ = '\0';
        }
        *ss++ = NULL;   /* terminate */

        entry = (void (*)(bd_t *, char *, char **, ulong))ntohl(hdr->ih_ep);
        (*entry)(kbd, cmdline, fwenv, top);
}
#endif


#if (CONFIG_COMMANDS & CFG_CMD_BOOTD)
int do_bootd (cmd_tbl_t *cmdtp, int flag, int argc, char *argv[])
{
        int rcode = 0;
#ifndef CFG_HUSH_PARSER
        if (run_command (getenv ("bootcmd"), flag) < 0) rcode = 1;
#else
        if (parse_string_outer(getenv("bootcmd"),
                FLAG_PARSE_SEMICOLON | FLAG_EXIT_FROM_LOOP) != 0 ) rcode = 1;
#endif
        return rcode;
}
#endif

#if (CONFIG_COMMANDS & CFG_CMD_IMI)
int do_iminfo ( cmd_tbl_t *cmdtp, int flag, int argc, char *argv[])
{
        int     arg;
        ulong   addr;
        int     rcode=0;

        if (argc < 2) {
                return image_info (load_addr);
        }

        for (arg=1; arg <argc; ++arg) {
                addr = simple_strtoul(argv[arg], NULL, 16);
                if (image_info (addr) != 0) rcode = 1;
        }
        return rcode;
}

static int image_info (ulong addr)
{
        ulong   data, len, checksum;
        image_header_t *hdr = &header;

        printf ("\n## Checking Image at %08lx ...\n", addr);

        /* Copy header so we can blank CRC field for re-calculation */
        memmove (&header, (char *)addr, sizeof(image_header_t));

        if (ntohl(hdr->ih_magic) != IH_MAGIC) {
                printf ("   Bad Magic Number\n");
```

```
                return 1;
        }

        data = (ulong)&header;
        len  = sizeof(image_header_t);

        checksum = ntohl(hdr->ih_hcrc);
        hdr->ih_hcrc = 0;

        if (crc32 (0, (char *)data, len) != checksum) {
                printf ("   Bad Header Checksum\n");
                return 1;
        }

        /* for multi-file images we need the data part, too */
        print_image_hdr ((image_header_t *)addr);

        data = addr + sizeof(image_header_t);
        len  = ntohl(hdr->ih_size);

        printf ("   Verifying Checksum ... ");
        if (crc32 (0, (char *)data, len) != ntohl(hdr->ih_dcrc)) {
                printf ("   Bad Data CRC\n");
                return 1;
        }
        printf ("OK\n");
        return 0;
}
#endif  /* CFG_CMD_IMI */

void
print_image_hdr (image_header_t *hdr)
{
#if (CONFIG_COMMANDS & CFG_CMD_DATE) || defined(CONFIG_TIMESTAMP)
        time_t timestamp = (time_t)ntohl(hdr->ih_time);
        struct rtc_time tm;
#endif

        printf ("   Image Name:   %.*s\n", IH_NMLEN, hdr->ih_name);
#if (CONFIG_COMMANDS & CFG_CMD_DATE) || defined(CONFIG_TIMESTAMP)
        to_tm (timestamp, &tm);
        printf ("   Created:      %4d-%02d-%02d  %2d:%02d:%02d UTC\n",
                tm.tm_year, tm.tm_mon, tm.tm_mday,
                tm.tm_hour, tm.tm_min, tm.tm_sec);
#endif  /* CFG_CMD_DATE, CONFIG_TIMESTAMP */
        printf ("   Image Type:   "); print_type(hdr); printf ("\n");
        printf ("   Data Size:    %d Bytes = ", ntohl(hdr->ih_size));
        print_size (ntohl(hdr->ih_size), "\n");
        printf ("   Load Address: %08x\n", ntohl(hdr->ih_load));
        printf ("   Entry Point:  %08x\n", ntohl(hdr->ih_ep));

        if (hdr->ih_type == IH_TYPE_MULTI) {
                int i;
                ulong len;
                ulong *len_ptr = (ulong *)((ulong)hdr + sizeof(image_header_t));

                printf ("   Contents:\n");
                for (i=0; (len = ntohl(*len_ptr)); ++i, ++len_ptr) {
                        printf ("   Image %d: %8ld Bytes = ", i, len);
                        print_size (len, "\n");
                }
        }
}


static void
print_type (image_header_t *hdr)
{
        char *os, *arch, *type, *comp;

        switch (hdr->ih_os) {
        case IH_OS_INVALID:     os = "Invalid OS";              break;
        case IH_OS_NETBSD:      os = "NetBSD";                  break;
        case IH_OS_LINUX:       os = "Linux";                   break;
        case IH_OS_VXWORKS:     os = "VxWorks";                 break;
        case IH_OS_QNX:         os = "QNX";                     break;
```

```
        case IH_OS_U_BOOT:      os = "U-Boot";                  break;
        case IH_OS_RTEMS:       os = "RTEMS";                   break;
#ifdef CONFIG_ARTOS
        case IH_OS_ARTOS:       os = "ARTOS";                   break;
#endif
        default:                os = "Unknown OS";              break;
        }

        switch (hdr->ih_arch) {
        case IH_CPU_INVALID:    arch = "Invalid CPU";           break;
        case IH_CPU_ALPHA:      arch = "Alpha";                 break;
        case IH_CPU_ARM:        arch = "ARM";                   break;
        case IH_CPU_I386:       arch = "Intel x86";             break;
        case IH_CPU_IA64:       arch = "IA64";                  break;
        case IH_CPU_MIPS:       arch = "MIPS";                  break;
        case IH_CPU_MIPS64:     arch = "MIPS 64 Bit";           break;
        case IH_CPU_PPC:        arch = "PowerPC";               break;
        case IH_CPU_S390:       arch = "IBM S390";              break;
        case IH_CPU_SH:         arch = "SuperH";                break;
        case IH_CPU_SPARC:      arch = "SPARC";                 break;
        case IH_CPU_SPARC64:    arch = "SPARC 64 Bit";          break;
        default:                arch = "Unknown Architecture";  break;
        }

        switch (hdr->ih_type) {
        case IH_TYPE_INVALID:   type = "Invalid Image";         break;
        case IH_TYPE_STANDALONE:type = "Standalone Program";    break;
        case IH_TYPE_KERNEL:    type = "Kernel Image";          break;
        case IH_TYPE_RAMDISK:   type = "RAMDisk Image";         break;
        case IH_TYPE_MULTI:     type = "Multi-File Image";      break;
        case IH_TYPE_FIRMWARE:  type = "Firmware";              break;
        case IH_TYPE_SCRIPT:    type = "Script";                break;
        default:                type = "Unknown Image";         break;
        }

        switch (hdr->ih_comp) {
        case IH_COMP_NONE:      comp = "uncompressed";          break;
        case IH_COMP_GZIP:      comp = "gzip compressed";       break;
        case IH_COMP_BZIP2:     comp = "bzip2 compressed";      break;
        default:                comp = "unknown compression";   break;
        }

        printf ("%s %s %s (%s)", arch, os, type, comp);
}

#define ZALLOC_ALIGNMENT        16

static void *zalloc(void *x, unsigned items, unsigned size)
{
        void *p;

        size *= items;
        size = (size + ZALLOC_ALIGNMENT - 1) & ~(ZALLOC_ALIGNMENT - 1);

        p = malloc (size);

        return (p);
}

static void zfree(void *x, void *addr, unsigned nb)
{
        free (addr);
}

#define HEAD_CRC        2
#define EXTRA_FIELD     4
#define ORIG_NAME       8
#define COMMENT         0x10
#define RESERVED        0xe0

#define DEFLATED        8

int gunzip(void *dst, int dstlen, unsigned char *src, int *lenp)
{
        z_stream s;
        int r, i, flags;
```

```
        /* skip header */
        i = 10;
        flags = src[3];
        if (src[2] != DEFLATED || (flags & RESERVED) != 0) {
                printf ("Error: Bad gzipped data\n");
                return (-1);
        }
        if ((flags & EXTRA_FIELD) != 0)
                i = 12 + src[10] + (src[11] << 8);
        if ((flags & ORIG_NAME) != 0)
                while (src[i++] != 0)
                        ;
        if ((flags & COMMENT) != 0)
                while (src[i++] != 0)
                        ;
        if ((flags & HEAD_CRC) != 0)
                i += 2;
        if (i >= *lenp) {
                printf ("Error: gunzip out of data in header\n");
                return (-1);
        }

        s.zalloc = zalloc;
        s.zfree = zfree;
#if defined(CONFIG_HW_WATCHDOG) || defined(CONFIG_WATCHDOG)
        s.outcb = (cb_func)WATCHDOG_RESET;
#else
        s.outcb = Z_NULL;
#endif  /* CONFIG_HW_WATCHDOG */

        r = inflateInit2(&s, -MAX_WBITS);
        if (r != Z_OK) {
                printf ("Error: inflateInit2() returned %d\n", r);
                return (-1);
        }
        s.next_in = src + i;
        s.avail_in = *lenp - i;
        s.next_out = dst;
        s.avail_out = dstlen;
        r = inflate(&s, Z_FINISH);
        if (r != Z_OK && r != Z_STREAM_END) {
                printf ("Error: inflate() returned %d\n", r);
                return (-1);
        }
        *lenp = s.next_out - (unsigned char *) dst;
        inflateEnd(&s);

        return (0);
}

static void
do_bootm_rtems (cmd_tbl_t *cmdtp, int flag, int argc, char *argv[],
                ulong addr, ulong *len_ptr, int verify)
{
        DECLARE_GLOBAL_DATA_PTR;
        image_header_t *hdr = &header;
        void    (*entry_point)(bd_t *);

        entry_point = (void (*)(bd_t *)) hdr->ih_ep;

        printf ("## Transferring control to RTEMS (at address %08lx) ...\n",
                (ulong)entry_point);

        SHOW_BOOT_PROGRESS (15);

        /*
         * RTEMS Parameters:
         *   r3: ptr to board info data
         */

        (*entry_point ) ( gd->bd );
}

#if (CONFIG_COMMANDS & CFG_CMD_ELF)
static void
```

```
do_bootm_vxworks (cmd_tbl_t *cmdtp, int flag, int argc, char *argv[],
                  ulong addr, ulong *len_ptr, int verify)
{
        image_header_t *hdr = &header;
        char str[80];

        sprintf(str, "%x", hdr->ih_ep); /* write entry-point into string */
        setenv("loadaddr", str);
        do_bootvx(cmdtp, 0, 0, NULL);
}

static void
do_bootm_qnxelf (cmd_tbl_t *cmdtp, int flag, int argc, char *argv[],
                 ulong addr, ulong *len_ptr, int verify)
{
        image_header_t *hdr = &header;
        char *local_args[2];
        char str[16];

        sprintf(str, "%x", hdr->ih_ep); /* write entry-point into string */
        local_args[0] = argv[0];
        local_args[1] = str;    /* and provide it via the arguments */
        do_bootelf(cmdtp, 0, 2, local_args);
}
#endif /* CFG_CMD_ELF */
```