****
   18492 Thu Feb 12 20:33:26 2015
new/usr/src/uts/armv6/loader/fakeloader.c
unix: enable caches in locore
The loader should really be as simple as possible to be as small as
possible. It should configure the machine so that unix can make certain
assumptions but it should leave more complex initialization to unix.
****

```
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy of the CDDL is also available via the Internet at
 * http://www.illumos.org/license/CDDL.
 */

/*
 * Copyright (c) 2014 Joyent, Inc.  All rights reserved.
 * Copyright (c) 2015 Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
 */

#include "fakeloader.h"

#include <sys/types.h>
#include <sys/param.h>
#include <sys/elf.h>
#include <sys/atag.h>
#include <sys/sysmacros.h>
#include <sys/machparam.h>

#include <vm/pte.h>

/*
 * This is the stock ARM fake uniboot loader.
 *
 * Here's what we have to do:
 *   o Read the atag header and find the combined archive header
 *   o Determine the set of mappings we need to add for the following:
 *              - unix
 *              - boot_archive
 *              - atags
 *   o Enable unaligned access
 *   o Enable virtual memory
 *   o Enable the caches + virtual memory
 *
 * There are several important constraints that we have here:
 *
 *   o We cannot use any .data! Several loaders that come before us are broken
 *     and only provide us with the ability to map our .text and potentially our
 *     .bss. We should strive to avoid even that if we can.
 */

#ifdef  DEBUG
#define FAKELOAD_DPRINTF(x)     fakeload_puts(x)
#else
#define FAKELOAD_DPRINTF(x)
#endif  /* DEBUG */

/*
 * XXX ASSUMES WE HAVE Free memory following the boot archive
 */
static uintptr_t freemem;
static uintptr_t pt_arena;
```

```
static uintptr_t pt_arena_max;
static uint32_t *pt_addr;
static int nl2pages;

/* Simple copy routines */
void
bcopy(const void *s, void *d, size_t n)
{
        const char *src = s;
        char *dest = d;

        if (n == 0 || s == d)
                return;

        if (dest < src && dest + n < src) {
                /* dest overlaps with the start of src, copy forward */
                for (; n > 0; n--, src++, dest++)
                        *dest = *src;
        } else {
                /* src overlaps with start of dest or no overlap, copy rev */
                src += n - 1;
                dest += n - 1;
                for (; n > 0; n--, src--, dest--)
                        *dest = *src;
        }
}
```
______***unchanged_portion_omitted_***

```
void
fakeload_init(void *ident, void *ident2, void *atag)
{
        atag_header_t *hdr;
        atag_header_t *chain = (atag_header_t *)atag;
        const atag_initrd_t *initrd;
        atag_illumos_status_t *aisp;
        atag_illumos_mapping_t *aimp;
        uintptr_t unix_start;

        fakeload_backend_init();
        fakeload_puts("Hello from the loader\n");
        initrd = (atag_initrd_t *)atag_find(chain, ATAG_INITRD2);
        if (initrd == NULL)
                fakeload_panic("missing the initial ramdisk\n");

        /*
         * Create the status atag header and the initial mapping record for the
         * atags. We'll hold onto both of these.
         */
        fakeload_mkatags(chain);
        aisp = (atag_illumos_status_t *)atag_find(chain, ATAG_ILLUMOS_STATUS);
        if (aisp == NULL)
                fakeload_panic("can't find ATAG_ILLUMOS_STATUS");
        aimp = (atag_illumos_mapping_t *)atag_find(chain, ATAG_ILLUMOS_MAPPING);
        if (aimp == NULL)
                fakeload_panic("can't find ATAG_ILLUMOS_MAPPING");
        FAKELOAD_DPRINTF("created proto atags\n");

        fakeload_pt_arena_init(initrd);

        fakeload_selfmap(chain);

        /*
         * Map the boot archive and all of unix
         */
        unix_start = fakeload_archive_mappings(chain,
            (const void *)(uintptr_t)initrd->ai_start, aisp);
```

```
	FAKELOAD_DPRINTF("filled out unix and the archive’s mappings\n");

	/*
	 * Fill in the atag mapping header for the atags themselves. 1:1 map it.
	 */
	aimp->aim_paddr = (uintptr_t)chain & ~0xfff;
	aimp->aim_plen = atag_length(chain) & ~0xfff;
	aimp->aim_plen += 0x1000;
	aimp->aim_vaddr = aimp->aim_paddr;
	aimp->aim_vlen = aimp->aim_plen;
	aimp->aim_mapflags = PF_R | PF_W | PF_NORELOC;

	/*
	 * Let the backend add mappings
	 */
	fakeload_backend_addmaps(chain);

	/*
	 * Turn on unaligned access
	 */
	FAKELOAD_DPRINTF("turning on unaligned access\n");
	fakeload_unaligned_enable();
	FAKELOAD_DPRINTF("successfully enabled unaligned access\n");

	/*
	 * To turn on the MMU we need to do the following:
	 *  o Program all relevant CP15 registers
	 *  o Program 1st and 2nd level page tables
	 *  o Invalidate and Disable the I/D-cache
	 *  o Fill in the last bits of the ATAG_ILLUMOS_STATUS atag
	 *  o Turn on the MMU in SCTLR
	 *  o Jump to unix
	 */

	/* Last bits of the atag */
	aisp->ais_freemem = freemem;
	aisp->ais_version = 1;
	aisp->ais_ptbase = (uintptr_t)pt_addr;

	/*
	 * Our initial page table is a series of 1 MB sections. While we really
	 * should map 4k pages, for the moment we’re just going to map 1 MB
	 * regions, yay team!
	 */
	hdr = chain;
	FAKELOAD_DPRINTF("creating mappings\n");
	while (hdr != NULL) {
		if (hdr->ah_tag == ATAG_ILLUMOS_MAPPING)
			fakeload_create_map(pt_addr,
			    (atag_illumos_mapping_t *)hdr);
		hdr = atag_next(hdr);
	}

	/*
	 * Now that we’ve mapped everything, update the status atag.
	 */
	aisp->ais_freeused = freemem - aisp->ais_freemem;
	aisp->ais_pt_arena = pt_arena;
	aisp->ais_pt_arena_max = pt_arena_max;

	/* Cache disable */
	FAKELOAD_DPRINTF("Flushing and disabling caches\n");
	armv6_dcache_flush();
	armv6_dcache_disable();
	armv6_dcache_inval();
	armv6_icache_disable();
```

```
	armv6_icache_inval();

	/* Program the page tables */
	FAKELOAD_DPRINTF("programming cp15 regs\n");
	fakeload_pt_setup((uintptr_t)pt_addr);


	/* MMU Enable */
	FAKELOAD_DPRINTF("see you on the other side\n");
	fakeload_mmu_enable();

	FAKELOAD_DPRINTF("why helo thar\n");

	/* Renable caches */
	armv6_dcache_enable();
	armv6_icache_enable();

	/* we should never come back */
	fakeload_exec(ident, ident2, chain, unix_start);
	fakeload_panic("hit the end of the world\n");
}
```
______***unchanged_portion_omitted_***

```
**********************************************************
    5123 Thu Feb 12 20:33:27 2015
new/usr/src/uts/armv6/ml/glocore.s
unix: enable caches in locore
The loader should really be as simple as possible to be as small as
possible.  It should configure the machine so that unix can make certain
assumptions but it should leave more complex initialization to unix.
**********************************************************
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy of the CDDL is also available via the Internet at
 * http://www.illumos.org/license/CDDL.
 */

/*
 * Copyright 2013 (c) Joyent, Inc. All rights reserved.
 * Copyright (c) 2015 Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
 */

#include <sys/asm_linkage.h>
#include <sys/machparam.h>
#include <sys/cpu_asm.h>

#include "assym.h"

/*
 * Every story needs a beginning. This is ours.
 */

/*
 * Each of the different machines has its own locore.s to take care of getting
 * the machine specific setup done.  Just before jumping into fakebop the
 * first time, we call this machine specific code.
 */

/*
 * We are in a primordial world here. The loader is going to come along and
 * boot us at _start. As we've started the world, we also need to set up a
 * few things about us, for example our stack pointer. To help us out, it's
 * useful to remember what the loader set up for us:
 *
 * - unaligned access are allowed (A = 0, U = 1)
 * - virtual memory is enabled
 *   - we (unix) are mapped right were we want to be
 *   - a UART has been enabled & any memory mapped registers have been 1:1
 *     mapped
 *   - ATAGs have been updated to tell us what the mappings are
 * - I/D L1 caches have may be disabled
 * - I/D L1 caches have been enabled
 */

        /*
         * External globals
         */
        .globl  _locore_start
        .globl  mlsetup
        .globl  sysp
        .globl  bootops
        .globl  bootopsp
        .globl  t0
```

```
        .data
        .comm   t0stack, DEFAULTSTKSZ, 32
        .comm   t0, 4094, 32


/*
 * Recall that _start is the traditional entry point for an ELF binary.
 */
        ENTRY(_start)
        ldr     sp, =t0stack
        ldr     r4, =DEFAULTSTKSZ
        add     sp, r4
        bic     sp, sp, #0xff

        /*
         * establish bogus stacks for exceptional CPU states, our exception
         * code should never make use of these, and we want loud and violent
         * failure should we accidentally try.
         */
        cps     #(CPU_MODE_UND)
        mov     sp, #-1
        cps     #(CPU_MODE_ABT)
        mov     sp, #-1
        cps     #(CPU_MODE_FIQ)
        mov     sp, #-1
        cps     #(CPU_MODE_IRQ)
        mov     sp, #-1
        cps     #(CPU_MODE_SVC)

        /* Enable highvecs (moves the base of the exception vector) */
        mrc     p15, 0, r3, c1, c0, 0
        orr     r3, r3, #(1 << 13)
        mcr     p15, 0, r3, c1, c0, 0

        /*
         * Go ahead now and enable the L1 I/D caches.  (Involves
         * invalidating the caches and the TLB.)
         */
        mov     r4, #0
        mov     r5, #0
        mcr     p15, 0, r4, c7, c7, 0   /* invalidate caches */
        mcr     p15, 0, r4, c8, c7, 0   /* invalidate tlb */
        mcr     p15, 0, r5, c7, c10, 4  /* DSB */
        mrc     p15, 0, r4, c1, c0, 0
        orr     r4, #0x04       /* D-cache */
        orr     r4, #0x1000     /* I-cache */
        mcr     p15, 0, r4, c1, c0, 0

#endif /* ! codereview */
        /* invoke machine specific setup */
        bl      _mach_start

        bl      _fakebop_start
        SET_SIZE(_start)


#if defined(__lint)

/* ARGSUSED */
void
_locore_start(struct boot_syscalls *sysp, struct bootops *bop)
{}

#else   /* __lint */

        /*
```

```
         * We got here from _kobj_init() via exitto().  We have a few different
         * tasks that we need to take care of before we hop into mlsetup and
         * then main. We're never going back so we shouldn't feel compelled to
         * preserve any registers.
         *
         *  o Enable our I/D-caches
         *  o Save the boot syscalls and bootops for later
         *  o Set up our stack to be the real stack of t0stack.
         *  o Save t0 as curthread
         *  o Set up a struct REGS for mlsetup
         *  o Make sure that we're 8 byte aligned for the call
         */

        ENTRY(_locore_start)


        /*
         * We've been running in t0stack anyway, up to this point, but
         * _locore_start represents what is in effect a fresh start in the
         * real kernel -- We'll never return back through here.
         *
         * So reclaim those few bytes
         */
        ldr     sp, =t0stack
        ldr     r4, =(DEFAULTSTKSZ - REGSIZE)
        add     sp, r4
        bic     sp, sp, #0xff

        /*
         * Save flags and arguments for potential debugging
         */
        str     r0, [sp, #REGOFF_R0]
        str     r1, [sp, #REGOFF_R1]
        str     r2, [sp, #REGOFF_R2]
        str     r3, [sp, #REGOFF_R3]
        mrs     r4, CPSR
        str     r4, [sp, #REGOFF_CPSR]

        /*
         * Save back the bootops and boot_syscalls.
         */
        ldr     r2, =sysp
        str     r0, [r2]
        ldr     r2, =bootops
        str     r1, [r2]
        ldr     r2, =bootopsp
        ldr     r2, [r2]
        str     r1, [r2]

        /*
         * Set up our curthread pointer
         */
        ldr     r0, =t0
        mcr     p15, 0, r0, c13, c0, 4

        /*
         * Go ahead now and enable the L1 I/D caches.
         */
        mrc     p15, 0, r0, c1, c0, 0
        orr     r0, #0x04       /* D-cache */
        orr     r0, #0x1000     /* I-cache */
        mcr     p15, 0, r0, c1, c0, 0

        /*
         * mlsetup() takes the struct regs as an argument. main doesn't take
         * any and should never return. Currently, we have an 8-byte aligned
```

```
         * stack.  We want to push a zero frame pointer to terminate any
         * stack walking, but that would cause us to end up with only a
         * 4-byte aligned stack.  So, to keep things nice and correct, we
         * push a zero value twice - it's similar to a typical function
         * entry:
         *      push { r9, lr }
         */
        mov     r9,#0
        push    { r9 }          /* link register */
        push    { r9 }          /* frame pointer */
        mov     r0, sp
        bl      mlsetup
        bl      main
        /* NOTREACHED */
        ldr     r0,=__return_from_main
        ldr     r0,[r0]
        bl      panic
        SET_SIZE(_locore_start)
```

______***unchanged_portion_omitted_***