Porting the Linux Kernel to the Ajit Processor Platform

Neha Karanjkar
30 January 2017

Overview

- The kernel's memory layout

- A minimal bootloader

- The initial memory map in the Ajit system

- The Ajit_linux kernel

- Generating the 'kernel+bootloader' memory map

The kernel's memory layout

- If each user process has its own virtual address space, how can a user process access kernel routines ?

Convention followed by the Linux kernel (architecture independant)
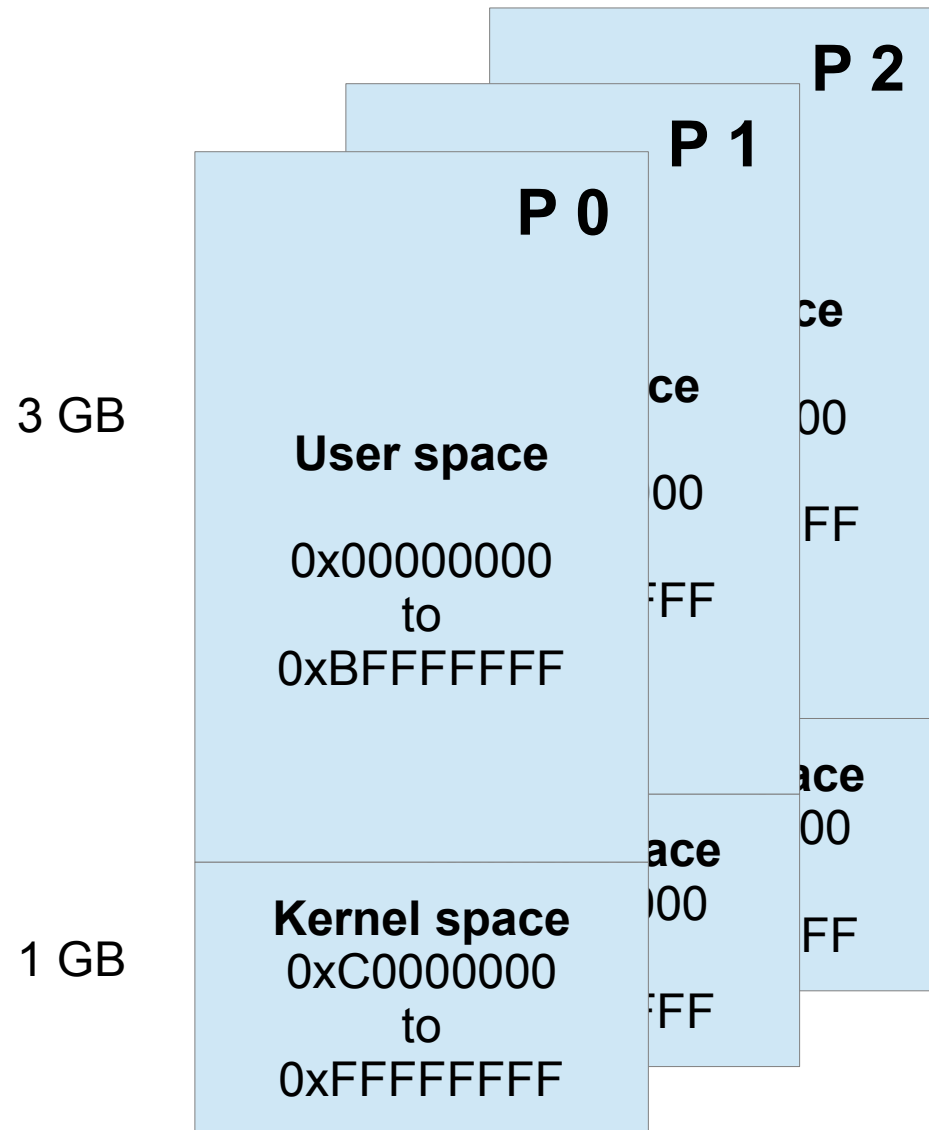
- A part of each user process' virtual address space is reserved for the kernel. The set of virtual addresses reserved for the kernel is fixed (0xC0000000 to 0xFFFFFFFF, total 1GB)

- Further, the kernel is always present (at a fixed location) in physical memory and never swapped out. The physical address of the kernel in memory is an architecture-specific parameter.

Convention followed by the Linux kernel (Sparc-v8 specific, fixed) :
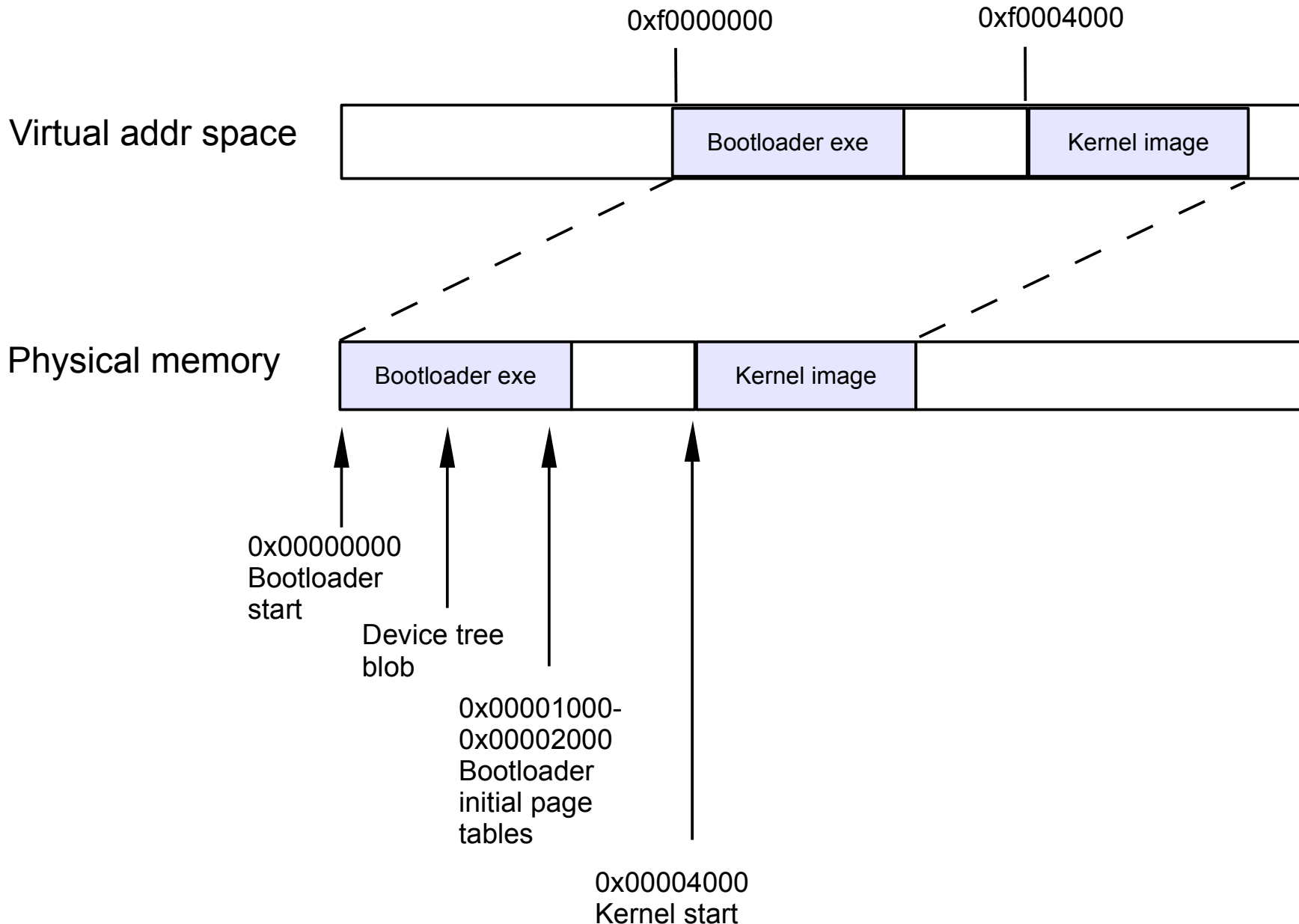
- The kernel symbols begin at virtual address 0xF0004000. The kernel assumes that this virtual address is mapped to physical address 0x00004000.

The kernel's memory layout

virtual memory layout (for 32b architectures)

P 2

P 1

P 0

3 GB

User space

0x00000000
to
0xBFFFFFFF

1 GB

Kernel space
0xC0000000
to
0xFFFFFFFF

The Ajit system memory layout

0xf0000000   0xf0004000

Virtual addr space

| | Bootloader exe | | Kernel image | |

Physical memory

| Bootloader exe | | Kernel image | |

0x00000000
Bootloader
start

Device tree
blob

0x00001000-
0x00002000
Bootloader
initial page
tables

0x00004000
Kernel start

A Minimal Bootloader for the Ajit Processor

- Kernel is already placed in memory at the correct physical address (0x00004000)

- The Bootloader needs to:

  - Initialize registers (zero out reg file, initialize stack ptr)

  - Setup initial page tables and enable the MMU

  - Populate a "romvec" structure through which hardware information is passed to the kernel

  - Jump to start of kernel (address 0xF0004000)

The Romvec Structure

- Important fields:

  - Pointer to a putchar() routine. The kernel printk's in-turn uses this putchar routine.

  - Information about available memory.

  - The nodeops. The nodeops consist of:

    - A device tree data-structure containing info about hardware (also called a device tree blob)

    - Routines to traverse this data structre.

  - The Ajit bootloader only passes a pointer to the device tree blob. The routines to traverse this blob are defined in the kernel. The routines are implemented as wrappers arount the standard flattened devices tree routines (in libfdt.h)

The Ajit_linux Kernel

- Derived from Linux 3.16.1

- Modifications limited to arch/sparc/

  - At kernel entry (head_32.s, setup_32.c)

  - Supplying machine-info to the kernel using device tree routines (arch/sparc/Ajit/ folder)

  - Low-level timer and interrupt-controller routines (Ajit_irq.c)

  - MMU routines (Ajit_srmmu.c)

- Serial device driver (Ajit_serial.c) added to /drivers/tty/serial

The Ajit_linux Kernel

- **Kernel entry:**

- The kernel's trap tables are located at the start of kernel address space. (/arch/sparc/kernel/ttable_32.S)

- The reset trap handler (located at 0xF0004000) jumps to the symbol "gokernel" (in /arch/sparc/kernel/head_32.S)

Generating the kernel memory map

- Detailed documentation at :

    /os/Ajit_linux/Documentation/README

- Script for the overall flow:

    /os/Ajit_linux/GeneratememMapForAjit.sh

- **Required:**

    – Buildroot

    See /os/Ajit_linux/Documentation/README

    – Sparc-v8 cross compiler toolchain (installed)

Generating the kernel memory map

The bootloader passes hardware info to the kernel through a device tree blob. The first step is to compile this device tree into assembly.

Device tree *(/arch/sparc/Ajit/device_tree/Ajit_device_tree.dts)*

|

compile_device_tree.sh
uses device tree compiler (dtc) + patch

↓

Ajit_device_tree.S

(The file Ajit_device_tree.S is then linked together with the bootloader code to generate a single bootloader executable)

Generating the kernel memory map

Kernel source
(Ajit_linux_3.16.1)

Buildroot configuration (Ajit_defconfig)

Linux kernel configuration
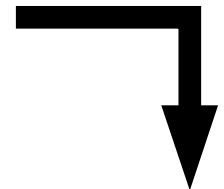(Ajit_minimal_linux_kernel.config)

Initramfs contents
rootfs.cpio (generated by buildroot)
+
Rootfs Overlay
(Ajit_buildroot_configs/board/rootfs_overlay/)

Buildroot

**Kernel
Executable
(vmlinux)**

Generating the kernel memory map

Bootloader source code
(AjitBootloader.C)

Ajit_device_tree.S

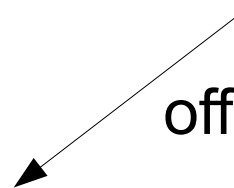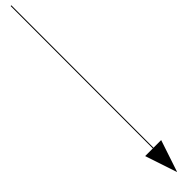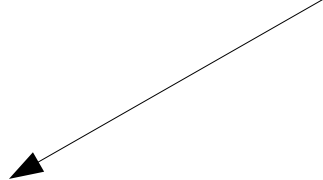**compileBootloader.py**

Bootloader executable

Kernel executable (vmlinux)

offset=0

offset= - 0xF0000000

vmlinuxToMemmap.py

MEMORY MAP FILE

TODO

- ~~Make the device tree blob + romvec a part of the bootloader~~ : DONE

- ~~Pass device-related information to the kernel solely through the device tree (not hard-wired into the kernel)~~: DONE

- Pass kernel arguments solely through the device tree (not hard-wired into the kernel)

- ~~Stack pointer initialization in the bootloader~~: DONE

- ~~Modify the bootloader to support a small-sized physical memory~~ : DONE

Other Contributors

- **Renju Boben** : Kernel modifications at the entry (romvec and device tree), Kernel compilation flow using Buildroot

- **Harshal Kalyane** : MMU validation, UART driver

- **Aniket Deshmukh** : Low level-timer interfacing, low-level UART interfacing, UART driver, Ajit_bootloader

- **Yeshwanth** : Low-level timer interfacing