The EFI Boot Stub

On the x86 and ARM platforms, a kernel zImage/bzImage can masquerade as a PE/COFF image, thereby convincing EFI firmware loaders to load it as an EFI executable. The code that modifies the bzImage header, along with the EFI-specific entry point that the firmware loader jumps to are collectively known as the "EFI boot stub", and live in arch/x86/boot/header.S and arch/x86/boot/compressed/eboot.c, respectively. For ARM the EFI stub is implemented in arch/arm/boot/compressed/efi-header.S and arch/arm/boot/compressed/efi-stub.c. EFI stub code that is shared between architectures is in drivers/firmware/efi/libstub.

For arm64, there is no compressed kernel support, so the Image itself masquerades as a PE/COFF image and the EFI stub is linked into the kernel. The arm64 EFI stub lives in arch/arm64/kernel/efi-entry.S and drivers/firmware/efi/libstub/arm64-stub.c.

By using the EFI boot stub it's possible to boot a Linux kernel without the use of a conventional EFI boot loader, such as grub or elilo. Since the EFI boot stub performs the jobs of a boot loader, in a certain sense it *IS* the boot loader.

The EFI boot stub is enabled with the CONFIG_EFI_STUB kernel option.

How to install bzImage.efi

The bzImage located in arch/x86/boot/bzImage must be copied to the EFI System Partition (ESP) and renamed with the extension ".efi". Without the extension the EFI firmware loader will refuse to execute it. It's not possible to execute bzImage.efi from the usual Linux file systems because EFI firmware doesn't have support for them. For ARM the arch/arm/boot/zImage should be copied to the system partition, and it may not need to be renamed. Similarly for arm64, arch/arm64/boot/Image should be copied but not necessarily renamed.

Passing kernel parameters from the EFI shell

Arguments to the kernel can be passed after bzImage.efi, e.g.:

```
fs0:> bzImage.efi console=ttyS0 root=/dev/sda4
```

The "initrd=" option

Like most boot loaders, the EFI stub allows the user to specify multiple initrd files using the "initrd=" option. This is the only EFI stub-specific command line parameter, everything else is passed to the kernel when it boots.

The path to the initrd file must be an absolute path from the beginning of the ESP, relative path names do not work. Also, the path is an EFI-style path and directory elements must be separated with backslashes (). For example, given the following directory layout:

```
fs0:>
      Kernels\
                      bzImage.efi
                      initrd-large.img

      Ramdisks\
                      initrd-small.img
                      initrd-medium.img
```

to boot with the initrd-large.img file if the current working directory is fs0:Kernels, the following command must be used:

```
fs0:\Kernels> bzImage.efi initrd=\Kernels\initrd-large.img
```

Notice how bzImage.efi can be specified with a relative path. That's because the image we're executing is interpreted by the EFI shell, which understands relative paths, whereas the rest of the command line is passed to bzImage.efi.

The "dtb=" option

For the ARM and arm64 architectures, a device tree must be provided to the kernel. Normally firmware shall supply the device tree via the EFI CONFIGURATION TABLE. However, the "dtb=" command line option can be used to override the firmware supplied device tree, or to supply one when firmware is unable to.

Please note: Firmware adds runtime configuration information to the device tree before booting the kernel. If dtb= is used to override the device tree, then any runtime data provided by firmware will be lost. The dtb= option should only be used either as a debug tool, or as a last resort when a device tree is not provided in the EFI CONFIGURATION TABLE.

"dtb=" is processed in the same manner as the "initrd=" option that is described above.