UEFI Bare Bones

From OSDev Wiki

Difficulty level

Medium

In this tutorial we will create a hard drive or ISO image containing a bare bones UEFI application for the x86_64 platform.

You are recommended to have read and fully understood the Bare Bones tutorial first. The UEFI page provides some background to the UEFI boot process and should also be consulted first.

This tutorial uses the header files and GUID definitions from the gnu-efi (https://sourceforge.net/projects/gnu-efi/) project, but does **not** use the gnu-efi build system, but rather the Mingw toolchain.

Contents

Prerequisites

You will need a GCC Cross-Compiler targeting the **x86_64-w64-mingw32** target, and the **gnu-efi** (https://sourceforge.net/projects/gnu-efi/) package to compile the actual kernel. To build the EFI filesystem image we use MTools and optionally **mkgpt** (http://www.tysos.org/redmine/projects/tysos/wiki/EFI) if you also want to create a hard disk image. To run under an emulator, we use **qemu-system-x86_64** and the **x64 OVMF firmware** (http://tianocore.sourceforge.net/wiki/OVMF) . If you wish to build a CD image, you will also need **xorriso**.

Under an apt-based system (e.g. Debian/Ubuntu) you can run

```
sudo apt-get install qemu binutils-mingw-w64 gcc-mingw-w64 xorriso mtools
wget http://www.tysos.org/files/efi/mkgpt-latest.tar.bz2
tar jxf mkgpt-latest.tar.bz2
```

```
cd mkgpt && ./configure && make && sudo make install && cd ..
```

and then separately download OVMF and extract the OVMF.fd file somewhere, and also gnu-efi.

Testing the emulator

Now is a good time to check the emulator is working successfully with the OVMF firmware.

```
qemu-system-x86_64 -L OVMF_dir/ -bios OVMF.fd
```

should launch qemu and dump you at a UEFI shell prompt.

hello.c

Create a file with the following:

```
#include <efi.h>
#include <efilib.h>

EFI_STATUS efi_main(EFI_HANDLE ImageHandle, EFI_SYSTEM_TABLE *SystemTable
{
    EFI_STATUS Status;
    EFI_INPUT_KEY Key;

    /* Store the system table for future use in other functions */
    ST = SystemTable;

    /* Say hi */
    Status = ST->ConOut->OutputString(ST->ConOut, L"Hello World\n\r");
    if (EFI_ERROR(Status))
        return Status;

    /* Now wait for a keystroke before continuing, otherwise your
       message will flash off the screen before you see it.

       First, we need to empty the console input buffer to flush
       out any keystrokes entered before this point */
    Status = ST->ConIn->Reset(ST->ConIn, FALSE);
    if (EFI_ERROR(Status))
        return Status;

    /* Now wait until a key becomes available.  This is a simple
       polling implementation.  You could try and use the WaitForKey
       event instead if you like */
    while ((Status = ST->ConIn->ReadKeyStroke(ST->ConIn, &Key)) == EFI_NC
```

```
    return Status;
}
```

gnu-efi/lib/data.c

We will also bring in the data.c file from the gnu-efi distribution, as this contains many predefined GUIDs for the various UEFI services. To avoid bloat and unnecessary dependencies on the rest of gnu-efi, you will need to edit it to remove the references to 'LibStubStriCmp', 'LibStubMetaiMatch' and 'LibStubStrLwrUpr' (simply make all the members of the LibStubUnicodeInterface structure be NULL).

gnu-efi/lib/lib.h

data.c includes this file. We copy it as-is to our source directory.

Building

To build, we use our cross-compiler:

```
x86_64-w64-mingw32-gcc -ffreestanding -Ipath/to/gnu-efi/inc -Ipath/to/gnu
x86_64-w64-mingw32-gcc -ffreestanding -Ipath/to/gnu-efi/inc -Ipath/to/gnu
x86_64-w64-mingw32-gcc -nostdlib -Wl,-dll -shared -Wl,--subsystem,10 -e e
```

Note here that '--subsystem 10' specifies an EFI application.

Creating the FAT image

Next, you will need to create a FAT filesystem image.

```
dd if=/dev/zero of=fat.img bs=1k count=1440
mformat -i fat.img -f 1440 ::
mmd -i fat.img ::/EFI
mmd -i fat.img ::/EFI/BOOT
mcopy -i fat.img BOOTX64.EFI ::/EFI/BOOT
```

Now, we can either use this as a USB stick image directly or embed it in HD image or CD image.

Running as a USB stick image

You can either write it directly to a USB stick and use in in a UEFI machine, or run it in qemu:

```
qemu-system-x86_64 -L OVMF_dir/ -bios OVMF.fd -usb -usbdevice disk::fat.i
```

Creating and running the HD image

The HD image is a disk image in the GPT format, with the FAT image specially identified as a 'EFI System Partition'.

```
mkgpt -o hdimage.bin --image-size 4096 --part fat.img --type system
qemu-system-x86_64 -L OVMF_dir/ -bios OVMF.fd -hda hdimage.bin
```

Creating and running the CD image

The iso image is a standard ISO9660 image which contains our FAT image as a file. A special El Torito option then points EFI aware systems to this image to be loaded. You can either burn the CD image to a CD and run it in a UEFI machine, or run it in qemu:

```
mkdir iso
cp fat.img iso
xorriso -as mkisofs -R -f -e fat.img -no-emul-boot -o cdimage.iso iso
qemu-system-x86_64 -L OVMF_dir/ -bios OVMF.fd -cdrom cdimage.iso
```

What to do next?

You may want to try using some more of the EFI boot services, e.g. to read more files from your FAT image, manage memory etc (see the UEFI Specifications (http://www.uefi.org/specifications) page for further documentation of this).

Common problems

Some UEFI hardware implementations require that the FAT image is in the FAT32 format (rather than FAT12 or FAT16). OVMF does not have this limitation, so you will not see such a problem in qemu. The minimum size of a FAT32 filesystem is, however, around 32 MiB so you will need to generate a much bigger image and pass the '-F' option to mformat.

See also

UEFI

Retrieved from "http://wiki.osdev.org/index.php?title=UEFI_Bare_Bones&oldid=17466"
Categories: Level 2 Tutorials | Bare bones tutorials

- This page was last modified on 13 January 2015, at 08:54.
- This page has been accessed 1,445 times.