presented by

Google

Extending EDK2 Functionalities to GNU-EFI

UEFI Fall 2023 Developers Conference & Plugfest
October 9-12, 2023
Presented by Mikolaj Lisik (Google)

Agenda

- Introduction
- What is GNU-EFI?
- Initial Attempts Using It
- Understanding the Internal Workings of GNU-EFI
- Now that We Know This - How Do We Add New Functionality to It
- Questions

What is GNU-EFI?

Why Building EFI Apps on Linux is Problematic

- EDK2 uses the PE32+ ABI
- GCC builds its targets as ELF binaries
- They are incompatible

The Solution - GNU-EFI

GNU-EFI serves as a bridge allowing to compile binaries that are compatible with UEFI by using the gcc compiler

Source code: https://sourceforge.net/projects/gnu-efi/

Initial Attempts at Using GNU-EFI

How Would We Use It?

Calling Any Arbitrary UEFI Function

The **libefi.a** has wrappers for the most common UEFI functions, but you might need to call something not covered. For completeness, it provides:

```
uefi_call_wrapper(func, numarg, ...);
```

Source - wiki.osdev.org/GNU-EFI

How Would We Use It?

Lets try it with:

```
RETURN_STATUS
EFIAPI
MemEncryptSevClearPageEncMask (
  IN PHYSICAL_ADDRESS   Cr3BaseAddress,
  IN PHYSICAL_ADDRESS   BaseAddress,
  IN UINTN              NumPages
  );
```

How Would We Use It?

```
uefi_call_wrapper(
      MemEncryptSevClearPageEncMask,
      3,
      Cr3BaseAddress,
      BaseAddress,
      NumPages);
```

How Would We Use It?

```
uefi_call_wrapper(
        MemEncryptSevClearPageEncMask,
        3,
        Cr3BaseAddress,
        BaseAddress,
        NumPages);
```

And it fails…

How Does GNU-EFI Implement Various Parts of EDK2?

How Does GNU-EFI Implement Various Parts of EDK2

- The types
- The function calls
- The UEFI API calls

The Types

How does GNU/EFI implement the types used internally by EFI?

The Types

They are copied…

The Types

Example from gnu_efi/inc/efidef.h:

```
//
//  Memory
//

typedef  UINT64                 EFI_PHYSICAL_ADDRESS;
typedef  UINT64                 EFI_VIRTUAL_ADDRESS;

typedef  enum {
      AllocateAnyPages,
      AllocateMaxAddress,
      AllocateAddress,
      MaxAllocateType
} EFI_ALLOCATE_TYPE;
```

The Function Calls

Most functions are…

The Function Calls

Most functions are… also copied.

The Function Calls

Example from gnu_efi/lib/misc.c:

```
VOID *
AllocatePool (
    IN UINTN                    Size
    )
{
    EFI_STATUS                  Status;
    VOID                        *p;

    Status = uefi_call_wrapper(BS->AllocatePool, 3, PoolAllocationType, Size, &p);
    if (EFI_ERROR(Status)) {
        DEBUG((D_ERROR, "AllocatePool: out of pool  %x\n", Status));
        p = NULL;
    }
    return p;
}
```

The Service Calls

They are not copied!

The Service Calls - Looking Back at osdev.org

For example, the "Print" function used in our main.c and which accepts printf compatible arguments, is under the hood nothing else than a call to:

```
uefi_call_wrapper(ST->ConOut->OutputString, 2, ST->ConOut, buffer);
```

The biggest advantage of 'uefi_call_wrapper' is that doesn't matter what ABI your gcc is using, it will always correctly translate that into UEFI ABI.

```
ST->ConOut->OutputString(ST->ConOut, buffer);
```

Source - wiki.osdev.org/GNU-EFI

The Service Calls

This is precisely what the uefi_call_wrapper is for.

The Service Calls

Taken from gnu_efi/inc/efiapi.h:

```
//
// EFI Boot Services Table
//

#define EFI_BOOT_SERVICES_SIGNATURE     0x56524553544f4f42
#define EFI_BOOT_SERVICES_REVISION      (EFI_SPECIFICATION_MAJOR_REVISION<<16) | (EFI_SPECIFICATION_MINOR_REVISION)

typedef struct _EFI_BOOT_SERVICES {

    EFI_TABLE_HEADER                Hdr;

    //
    // Task priority functions
    //

    EFI_RAISE_TPL                   RaiseTPL;
    EFI_RESTORE_TPL                 RestoreTPL;
```

The Service Calls

Taken from gnu_efi/inc/x86_64/efibind.h:

```
/* main wrapper (va_num ignored) */
#define uefi_call_wrapper(func, va_num, ...)
\
  __VA_ARG_NSUFFIX__(_cast64_efi_call,
__VA_ARGS__) (func , ##__VA_ARGS__)
```

The Service Calls

Taken from gnu_efi/inc/x86_64/efibind.h:

```
#define _cast64_efi_call2(f,a1,a2) \
  efi_call2(f, (UINT64)(a1), (UINT64)(a2))
#define _cast64_efi_call3(f,a1,a2,a3) \
  efi_call3(f, (UINT64)(a1), (UINT64)(a2), (UINT64)(a3))
#define _cast64_efi_call4(f,a1,a2,a3,a4) \
  efi_call4(f, (UINT64)(a1), (UINT64)(a2), (UINT64)(a3),
(UINT64)(a4))
```

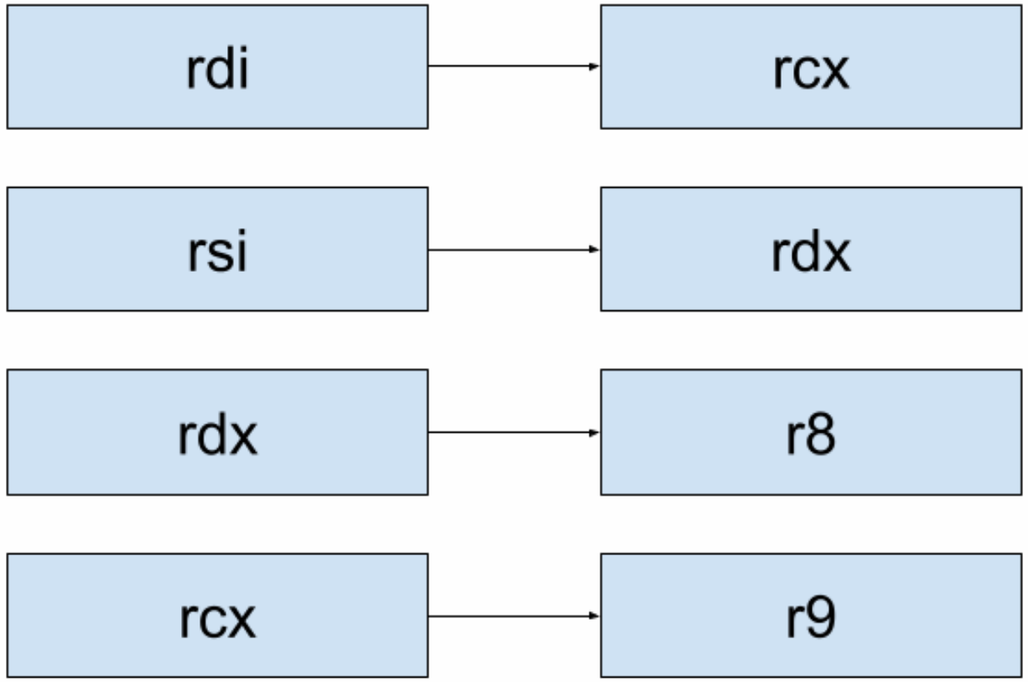
The Service Calls

Taken from gnu_efi/lib/x86_64/efi_stub.S:

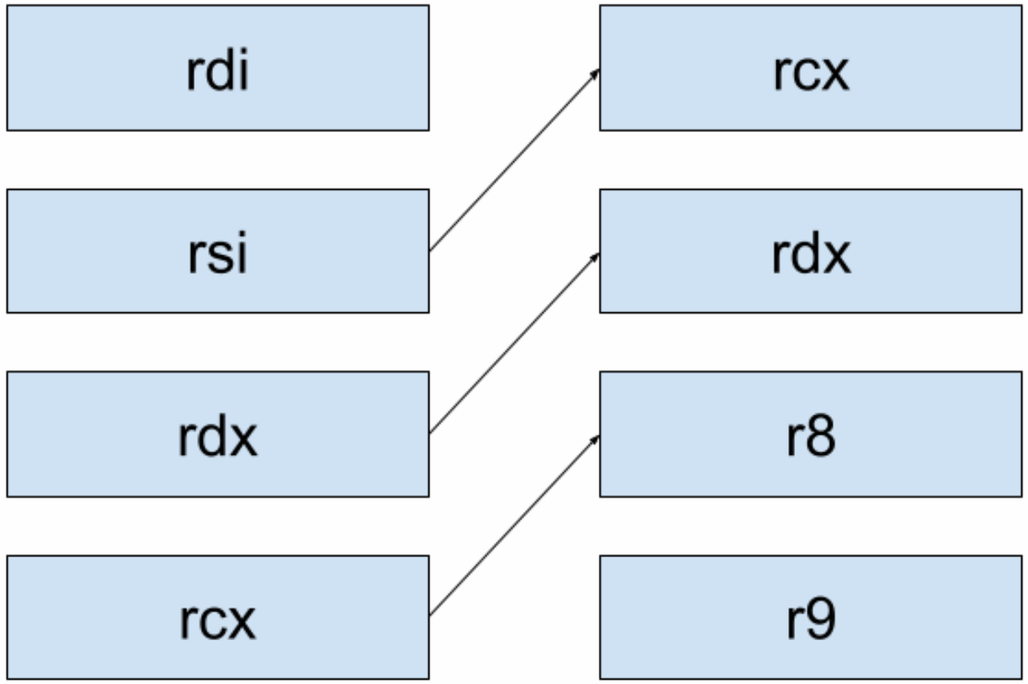
```
ENTRY(efi_call3)
        subq $40, %rsp
        mov  %rcx, %r8
        /* mov %rdx, %rdx */
        mov  %rsi, %rcx
        call *%rdi
        addq $40, %rsp

        ret
```

The Service Calls - Register Value Conversions



The Service Calls - Register Value Conversions



The Service Calls

Taken from gnu_efi/lib/x86_64/efi_stub.S:

```
ENTRY(efi_call3)
        subq $40, %rsp
        mov  %rcx, %r8
        /* mov %rdx, %rdx */
        mov  %rsi, %rcx
        call *%rdi
        addq $40, %rsp

        ret
```

Porting Code to GNU-EFI

Porting Code to GNU-EFI

- The types and functions can be used as long as they have already been ported to GNU-EFI.
- BS and RT service calls can be used as long as they have been ported to efiapi.h. Porting new services is trivial.

Questions

Thanks for attending the UEFI Fall 2023 Developers Conference & Plugfest

For more information on UEFI Forum and UEFI Specifications, visit http://www.uefi.org

presented by

Google