Programming UEFI for dummies

Or

What I have learned while tweaking FreePascal to output UEFI binaries
UEFI

• Unified Extensible Firmware Interface
• Specification that define an abstract common interface over firmware
• For short : BIOS replacement
What I will discuss?

- Quick overview of existing UEFI toolchains
- Structure of UEFI executable files
- Structure of UEFI APIs
- Overview of features exposed by UEFI APIs
- Protocols
- Bonus feature...
- What’s next?
Disclaimer notice

- While very important, this presentation will not discuss any security issues of UEFI
- I assume SecureBoot is disabled to use what is presented here
Existing toolchains

- Mainly two stacks
  - TianoCore EDK II
  - GNU-EFI

- From what I read
  - Tedious setup process (more than one package)
  - GNU-EFI is supposed **simpler** to use (not simple ;-)"
    - Do not require a full cross compiler
Binary structure of UEFI application

• Portable Executable binaries (PE32 or PE32+ for x86* and ARM CPUs)
• With a special subsystem code to recognize an UEFI application from a Windows binary
  - Applications
    • EFI_APP (11) : bootloader, baremetal applications...
  - drivers
    • EFI_BOOT (12) : filesystem...
    • EFI_RUN (13) : available to OS at runtime
UEFI application entry point

• EFI_MAIN( imageHandle: EFI_HANDLE; systemTable : PEFI_SYSTEM_TABLE): EFI_STATUS;

• Same calling convention as the corresponding Windows target

• CPU already in protected mode with flat memory model
  - On 64 bits, already in long mode
  - But only one CPU core initialized
Overview of EFI_SYSTEM_TABLE

• Access to Input/output/error console
• Access to Configuration tables (ACPI tables...)
• 2 substructures:
  – RuntimesServices
  – BootServices
Features available from EFI_SYSTEM_TABLE

- Memory allocation
- Getting memory map (useful for OS)
- Console Input/Output
- File system access
- GUI with mouse support*
- Network access*
- ...

* Depends on your machine firmware implementation
UEFI protocols

• More or less interfaces with a set of related methods and fields

• Identified by a GUID

  For example:
  - SIMPLE_TEXT_OUTPUT_PROTOCOL
  - SIMPLE_TEXT_INPUT_PROTOCOL
  - GRAPHICS OUTPUT PROTOCOL (GOP)

• Some are at specific SYSTEM_TABLE position
Finding a protocol

• Find the others by GUIDs or handle
• HandleProtocol :
  – Check is a handle support a specific protocol
• LocateProtocol
  – Finds first handle that support a specified protocol
UEFI programming tips

• ExitBootServices
  - Last step before giving control to OS
• BootServices → SetWatchdogTimer(0,0,0,NULL)
  - If you need more than 5 minutes...
Ready to write your first UEFI application
One more thing...
Introducing the easiest way to write UEFI applications ;-)*

Freepascal

• One tool
  - Use Freepascal internal assembler and linker
  - No external dependencies required
  - Easy to setup
    • Even under Haiku ;-
  - Still work in progress

* Well, that’s the plan...
Hello world comparison

- GNU-EFI
  ```c
  #include <efi.h>
  #include <efilib.h>
  EFI_STATUS
  efi_main (EFI_HANDLE image, EFI_SYSTEM_TABLE *systab)
  {
      SIMPLE_TEXT_OUTPUT_INTERFACE *conout;
      conout = systab->ConOut;
      InitializeLib(image, systab);
      uefi_call_wrapper(conout->OutputString, 2, conout, L"Hello World!
      return EFI_SUCCESS;
  }
  ```

- Freepascal
  ```pascal
  Program hello;
  Begin
      WriteLn(‘Hello World!’);
  End.
  ```

- TianoCore
  ```c
  /** @file
      Brief Description of UEFI MyHelloWorld
      Detailed Description of UEFI MyHelloWorld
      Copyright for UEFI MyHelloWorld
      License for UEFI MyHelloWorld
      **/ 
      #include <Uefi.h>
      #include <Library/UefiApplicationEntryPoint.h>
      #include <Library/UefiLib.h>
      /**
       as the real entry point for the application.
      
      @param[in] ImageHandle  The firmware allocated handle for the EFI image.
      @param[in] SystemTable  A pointer to the EFI System Table.
      
      @retval EFI_SUCCESS      The entry point is executed successfully.
      @retval other            Some error occurs when executing this entry point.
      **/ 
      EFI_STATUS
      EFI_API
      UefiMain ( 
        IN EFI_HANDLE        ImageHandle,
        IN EFI_SYSTEM_TABLE  *SystemTable
      ) 
      { 
          Print(L"Hello World \n"); 
          return EFI_SUCCESS;
      }
  ```
What’s next?

- Full Runtime Library support under UEFI (RTL)
- Binding for all UEFI protocols?
  - Help is welcome!
- Maybe FreePascal as an UEFI Application? ;-)
Links

• Where everything start for me
  – http://blog.theincredibleholk.org/blog/2013/11/18/booting-to-rust/

• Freepascal UEFI target
  – http://wiki.freepascal.org/UEFI
  – https://svn.freepascal.org/cgi-bin/viewvc.cgi/branches/olivier/uefi/

• Setup GNU-EFI
  – https://mjg59.dreamwidth.org/18773.html
  – https://sourceforge.net/projects/gnu-efi/

• TianoCore
The end...

Happy UEFI hacking!

• Questions?