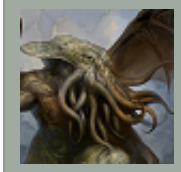
Uefi.inc

From OSDev Wiki



Ph'nglui mglw'nafh Cthulhu R'lyeh wgah'nagl fhtagn!

The content of this article or section is Lovecraftian.

`uefi.inc` a library for UEFI written in assembly for fasm. It's main goal was small code, and to achieve that it gives only limited access to UEFI. It was designed to fulfill all the needs of a boot loader, namely

- provide user input interface
- exactly one way to load and save sectors
- framebuffer services for output

It is not intend to be an universal UEFI library.

Contents

Hello World from UEFI

This shows how to import and use the library.

```
format pe64 dll efi
entry main

section '.text' code executable readable

include 'uefi.inc'

main:
    ; initialize UEFI library
```

```
    InitializeLib
    jc @f

    ; call uefi function to print to screen
    uefi_call_wrapper ConOut, OutputString, ConOut, _hello

@@: mov eax, EFI_SUCCESS
    retn

section '.data' data readable writeable

_hello                                  du 'Hello World',13,10,0

section '.reloc' fixups data discardable
```

Case studies: usage in a boot loader

Please note that these are only tutorials. They focus on a specific topic only. They do not allocate buffers they use for example. You'll have to write that part on your own. They were written for those who wants to roll their own UEFI boot loader in assembly. I hope it will save them some sleepless nights and headaches.

Reading a character from keyboard

```
@@:             uefi_call_wrapper       ConIn, ReadKeyStroke, Co
                cmp                     dword [key.scancode], 0
                jz                      @b

;data
key:
key.scancode:   dw                      0
key.unicode:    du                      0
```

Detecting Memory Map

```
                mov                     dword [memmapdescsize],
                uefi_call_wrapper       BootServices, GetMemoryM
                cmp                     dword [memmapdescsize],
                jnz                     @f
                mov                     dword [memmapdescsize],
@@:             clc
                cmp                     rax, EFI_SUCCESS
                je                      @f
                stc
@@:
```

```
;data
memmapsize:      dq                      MEMMAP_BUFFSIZE
memmapkey:       dq                      0
memmapdescsize:  dq                      0
memmapdescver:   dq                      0
memmapbuff:      rb                      MEMMAP_BUFFSIZE
```

Note that memmapkey is needed for ExitBootServices. If you use memory allocation or free, you'll have to query again map to get a valid key.

Iterating on disks

```
                ; query device handles
                mov                     dword [tmp], DEVICEHANDL
                uefi_call_wrapper       BootServices, LocateHand
                cmp                     rax, EFI_SUCCESS
                jne                     .error

                ; iterate on handles
                mov                     r15, devicehandle_buff
                mov                     r14, qword [tmp]
.nexthandle:    mov                     rcx, qword [r15]
                or                      rcx, rcx
                jz                      .end
                ; open handle
                push                    r14
                push                    r15
                uefi_call_wrapper       BootServices, OpenProtoc
                pop                     r15
                pop                     r14
                cmp                     rax, EFI_SUCCESS
                jne                     .skipdev
                ; save results
                mov                     rax, qword [tmp]
                mov                     qword [bootdiskblkio], r
                ; FIXME: do some checks here to decide if it's y
                ; I load GPT from each device and look for a spe
                ; cmp ismydevice, 1
                ; je .end

.skipdev:       xor                     rax, rax
                mov                     qword [bootdiskblkio], r
                pop                     r15
                pop                     r14
                add                     r15, 8
                sub                     r14, 8
```

```
                js                      .error
                jnz                     .nexthandle

.end:           clc
                ret

.error:         stc
                ret

;data
tmp:            dq                      0
blkiouuid:      db                      EFI_BLOCK_IO_PROTOCOL_UU
bootdiskblkio:  dq                      0
```

Loading a sector from device

```
                mov                     rax, qword [bootdiskblki
                stc
                or                      rax, rax
                jz                      @f
                mov                     rbx, rax
                mov                     rcx, rax
                mov                     rax, qword [rax+EFI_BLOC
                or                      rax, rax
                jz                      @f
                mov                     edx, dword [rax+EFI_BLOC
                or                      edx, edx
                jz                      @f
                xor                     rax, rax
                uefi_call_wrapper       rbx, EFI_BLOCK_IO_PROTOC
                stc
                cmp                     rax, EFI_SUCCESS
                jne                     @f
                clc
@@:

;data
lba:            dq                      0
sizeinbytes:    dq                      0
diskbuff:       rb                      DISK_BUFFSIZE
```

Get sorted list of available video modes

```
                ; installation check on GOP. This must return bu
```

```
                xor                     rax, rax
                mov                     qword [tmp], rax
                uefi_call_wrapper       BootServices, LocateHand
                cmp                     rax, EFI_BUFFER_TOO_SMAL
                jne                     .error
                ; query available video modes
                mov                     dword [tmp], MODE_BUFFSI
                uefi_call_wrapper       BootServices, LocateHand
                cmp                     rax, EFI_SUCCESS
                jne                     .error
                ; iterate on each handle, store sorted result to
                mov                     rsi, qword [tmpbuff]
                lea                     rdi, [gopmodes]
.getnextinfo:   lodsq
                or                      rax, rax
                jz                      .nomore
                mov                     qword [tmp], rax
                mov                     qword [rdi], rax
                push                    rsi
                push                    rdi
                uefi_call_wrapper       BootServices, HandleProt
                pop                     rdi
                pop                     rsi
                cmp                     rax, EFI_SUCCESS
                jne                     .error
                ; iterate on each mode for a handle
                ; get number of modes
                mov                     rcx, qword [gopinterface
                mov                     r10, qword [rcx + EFI_GR
                mov                     r10, qword [r10 + EFI_GR
                xor                     rdx, rdx
.nextmode:      mov                     rcx, qword [gopinterface
                mov                     qword [rdi+8], rcx
                push                    rsi
                push                    rdi
                mov                     dword [tmp], 8
                lea                     r8, [tmp]
                lea                     r9, [gopinfo]
                push                    rcx
                push                    rdx
                push                    r10
                uefi_call_wrapper       rcx, EFI_GRAPHICS_OUTPUT
                pop                     r10
                pop                     rdx
                pop                     rcx
                pop                     rdi
                pop                     rsi
                cmp                     rax, EFI_SUCCESS
                jne                     .error
```

```
                ; save info to gopmodes
                mov                     r9, qword [gopinfo]
                mov                     eax, dword [r9+EFI_GRAPH
                mov                     word [rdi+16], ax
                mov                     eax, dword [r9+EFI_GRAPH
                mov                     word [rdi+18], ax
                mov                     eax, dword [r9+EFI_GRAPH
                mov                     word [rdi+20], ax
                mov                     ax, word [r9+EFI_GRAPHIC
                mov                     word [rdi+22], ax
                mov                     dword [rdi+24], edx
                ;bubble up mode
                mov                     rbx, rdi
.bubble:                mov                             ax, word [rdi+16
                cmp                     word [rdi+16-32], ax
                ja                      .ok2
                jb                      .switch
                mov                     ax, word [rdi+18]
                cmp                     word [rdi+18-32], ax
                jae                     @f
.switch:                mov                             rax, qword [rdi]
                xchg                    qword [rdi-32], rax
                mov                     qword [rdi], rax
                mov                     rax, qword [rdi+8]
                xchg                    qword [rdi-32+8], rax
                mov                     qword [rdi+8], rax
                mov                     rax, qword [rdi+16]
                xchg                    qword [rdi-32+16], rax
                mov                     qword [rdi+16], rax
                mov                     rax, qword [rdi+24]
                xchg                    qword [rdi-32+24], rax
                mov                     qword [rdi+24], rax
                sub                     rdi, 32
                cmp                     rdi, gopmodes
                jae                     .bubble
@@:             mov                     rdi, rbx
                add                     rdi, 32
.loop:          inc                     rdx
                cmp                     edx, r10d
                jb                      .nextmode
                jmp                     .getnextinfo
.error:         xor                     rax, rax
                mov                     qword [gopinterface], ra
                stc
.nomore:

;data
tmp:            dq                      0
tmpbuff:        dq                      0
```

```
gopuuid:        db                          EFI_GRAPHICS_OUTPUT_PROT
gop_handle:     dq                          0
gopinterface:   dq                          0
gopinfo:        dq                          0
;output, sorted list of 32 byte entries
;  8 bytes handle
;  8 bytes interface
;  2 bytes horizontal resolution
;  2 bytes vertical resolution
;  2 bytes scanline
;  2 bytes pixelformat
;  8 bytes mode number
gopmodes:       rb                          MODE_BUFFSIZE
```

To access framebuffer, you'll have to get it's address. This code returns it in rbx, and buffer's size in rcx:

```
                mov                         rax, qword [gopinterface
                mov                         rax, qword [rax + EFI_GR
                mov                         rbx, qword [rax + EFI_GR
                mov                         rcx, qword [rax + EFI_GR
```

The include file

And finally the library that makes the magic alive:

```
;******************************************************************
;*
;*          UEFI library for fasm by bzt, Public Domain
;*
;******************************************************************

; include x86asm.net's efi.inc
;include 'efi.inc'
struc int8 {
   . db ?
}
struc int16 {
  align 2
   . dw ?
}
struc int32 {
  align 4
   . dd ?
}
struc int64 {
```

```
  align 8
  . dq ?
}
struc intn {
  align 8
  . dq ?
}
struc dptr {
  align 8
  . dq ?
}

;symbols

EFIERR = 0x8000000000000000
EFI_SUCCESS                     = 0
EFI_LOAD_ERROR                  = EFIERR or 1
EFI_INVALID_PARAMETER           = EFIERR or 2
EFI_UNSUPPORTED                 = EFIERR or 3
EFI_BAD_BUFFER_SIZE             = EFIERR or 4
EFI_BUFFER_TOO_SMALL            = EFIERR or 5
EFI_NOT_READY                   = EFIERR or 6
EFI_DEVICE_ERROR                = EFIERR or 7
EFI_WRITE_PROTECTED             = EFIERR or 8
EFI_OUT_OF_RESOURCES            = EFIERR or 9
EFI_VOLUME_CORRUPTED            = EFIERR or 10
EFI_VOLUME_FULL                 = EFIERR or 11
EFI_NO_MEDIA                    = EFIERR or 12
EFI_MEDIA_CHANGED               = EFIERR or 13
EFI_NOT_FOUND                   = EFIERR or 14
EFI_ACCESS_DENIED               = EFIERR or 15
EFI_NO_RESPONSE                 = EFIERR or 16
EFI_NO_MAPPING                  = EFIERR or 17
EFI_TIMEOUT                     = EFIERR or 18
EFI_NOT_STARTED                 = EFIERR or 19
EFI_ALREADY_STARTED             = EFIERR or 20
EFI_ABORTED                     = EFIERR or 21
EFI_ICMP_ERROR                  = EFIERR or 22
EFI_TFTP_ERROR                  = EFIERR or 23
EFI_PROTOCOL_ERROR              = EFIERR or 24

;helper macro for definition of relative structure member offset

macro struct name
{
  virtual at 0
    name name
  end virtual
}
```

```
;structures
EFI_SYSTEM_TABLE_SIGNATURE      equ     49h,42h,49h,20h,53h,59h,
struc EFI_TABLE_HEADER {
  .Signature              int64
  .Revision               int32
  .HeaderSize             int32
  .CRC32                  int32
  .Reserved               int32
}
struct EFI_TABLE_HEADER

struc EFI_SYSTEM_TABLE {
  .Hdr                    EFI_TABLE_HEADER
  .FirmwareVendor         dptr
  .FirmwareRevision       int32
  .ConsoleInHandle        dptr
  .ConIn                  dptr
  .ConsoleOutHandle       dptr
  .ConOut                 dptr
  .StandardErrorHandle    dptr
  .StdErr                 dptr
  .RuntimeServices        dptr
  .BootServices           dptr
  .NumberOfTableEntries   intn
  .ConfigurationTable     dptr
}
struct EFI_SYSTEM_TABLE

struc SIMPLE_TEXT_OUTPUT_INTERFACE {
  .Reset                  dptr
  .OutputString           dptr
  .TestString             dptr
  .QueryMode              dptr
  .SetMode                dptr
  .SetAttribute           dptr
  .ClearScreen            dptr
  .SetCursorPosition      dptr
  .EnableCursor           dptr
  .Mode                   dptr
}
struct SIMPLE_TEXT_OUTPUT_INTERFACE

;---include ends

struc SIMPLE_INPUT_INTERFACE {
  .Reset                  dptr
  .ReadKeyStroke          dptr
  .WaitForKey             dptr
}
```

```
struct SIMPLE_INPUT_INTERFACE

struc EFI_BOOT_SERVICES_TABLE {
  .Hdr                    EFI_TABLE_HEADER
  .RaisePriority          dptr
  .RestorePriority        dptr
  .AllocatePages          dptr
  .FreePages              dptr
  .GetMemoryMap           dptr
  .AllocatePool           dptr
  .FreePool               dptr
  .CreateEvent            dptr
  .SetTimer               dptr
  .WaitForEvent           dptr
  .SignalEvent            dptr
  .CloseEvent             dptr
  .CheckEvent             dptr
  .InstallProtocolInterface dptr
  .ReInstallProtocolInterface dptr
  .UnInstallProtocolInterface dptr
  .HandleProtocol         dptr
  .Void                   dptr
  .RegisterProtocolNotify dptr
  .LocateHandle           dptr
  .LocateDevicePath       dptr
  .InstallConfigurationTable dptr
  .ImageLoad              dptr
  .ImageStart             dptr
  .Exit                   dptr
  .ImageUnLoad            dptr
  .ExitBootServices       dptr
  .GetNextMonotonicCount dptr
  .Stall                  dptr
  .SetWatchdogTimer       dptr
  .ConnectController      dptr
  .DisConnectController   dptr
  .OpenProtocol           dptr
  .CloseProtocol          dptr
  .OpenProtocolInformation dptr
  .ProtocolsPerHandle     dptr
  .LocateHandleBuffer     dptr
  .LocateProtocol         dptr
  .InstallMultipleProtocolInterfaces dptr
  .UnInstallMultipleProtocolInterfaces dptr
  .CalculateCrc32         dptr
  .CopyMem                dptr
  .SetMem                 dptr
}
struct EFI_BOOT_SERVICES_TABLE

struc EFI_RUNTIME_SERVICES_TABLE {
```

```
  .Hdr                    EFI_TABLE_HEADER
  .GetTime                dptr
  .SetTime                dptr
  .GetWakeUpTime          dptr
  .SetWakeUpTime          dptr
  .SetVirtualAddressMap   dptr
  .ConvertPointer         dptr
  .GetVariable            dptr
  .GetNextVariableName    dptr
  .SetVariable            dptr
  .GetNextHighMonoCount   dptr
  .ResetSystem            dptr
}
struct EFI_RUNTIME_SERVICES_TABLE

struc EFI_TIME {
  .Year                   int16
  .Month                  int8
  .Day                    int8
  .Hour                   int8
  .Minute                 int8
  .Second                 int8
  .Pad1                   int8
  .Nanosecond             int32
  .TimeZone               int16
  .Daylight               int8
  .Pad2                   int8
  .sizeof                 rb 1
}
struct EFI_TIME

EFI_LOADED_IMAGE_PROTOCOL_UUID equ 0A1h,31h,1bh,5bh,62h,95h,0d2h
struc EFI_LOADED_IMAGE_PROTOCOL {
  .Revision               int32
  .ParentHandle           int64
  .SystemTable            dptr
  .DeviceHandle           int64
  .FilePath               dptr
  .Reserved               int64
  .LoadOptionsSize        int32
  .ImageBase              dptr
  .ImageSize              int64
  .ImageCodeType          int32
  .ImageDataType          int32
  .UnLoad                 dptr
}
struct EFI_LOADED_IMAGE_PROTOCOL

EFI_BLOCK_IO_PROTOCOL_UUID equ 21h,5bh,4eh,96h,59h,64h,0d2h,11h,
struc EFI_BLOCK_IO_PROTOCOL {
  .Revision               int64
```

```
  .Media                    dptr
  .Reset                    dptr
  .ReadBlocks               dptr
  .WriteBlocks              dptr
  .FlushBlocks              dptr
}
struct EFI_BLOCK_IO_PROTOCOL

struc EFI_BLOCK_IO_MEDIA {
  .MediaId                  int32
  .RemovableMedia           int8
  .MediaPresent             int8
  .LogicalPartition         int8
  .ReadOnly                 int8
  .WriteCaching             int8
  .BlockSize                int32
  .IoAlign                  int32
  .LastBlock                int64
}
struct EFI_BLOCK_IO_MEDIA

EFI_GRAPHICS_OUTPUT_PROTOCOL_UUID equ 0deh, 0a9h, 42h,90h,0dch,0
struc EFI_GRAPHICS_OUTPUT_PROTOCOL {
  .QueryMode                dptr
  .SetMode                  dptr
  .Blt                      dptr
  .Mode                     dptr
}
struct EFI_GRAPHICS_OUTPUT_PROTOCOL

struc EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE {
  .MaxMode                  int32
  .CurrentMode              int32
  .ModeInfo                 dptr
  .SizeOfModeInfo           intn
  .FrameBufferBase          dptr
  .FrameBufferSize          intn
}
struct EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE

struc EFI_GRAPHICS_OUTPUT_MODE_INFORMATION {
  .Version                  int32
  .HorizontalResolution     int32
  .VerticalResolution       int32
  .PixelFormat              int32
  .RedMask                  int32
  .GreenMask                int32
  .BlueMask                 int32
  .Reserved                 int32
  .PixelsPerScanline        int32
```

```
}
struct EFI_GRAPHICS_OUTPUT_MODE_INFORMATION
;---macros to make life easier---
;call it early, after entry point is the best
macro InitializeLib
{
                clc
                or                              rdx, rdx
                jz                              .badout
                cmp                             dword [rdx], 20494249h
                je                              @f
.badout:          xor                           rcx, rcx
                xor                             rdx, rdx
                stc
@@:             mov                             [efi_handler], rcx
                mov                             [efi_ptr], rdx
}

;invoke an UEFI function
macro uefi_call_wrapper                         interface,function,arg1,
{
numarg = 0
if ~ arg11 eq
 numarg = numarg + 1
 if ~ arg11 eq rdi
                mov                             rdi, arg11
 end if
end if
if ~ arg10 eq
 numarg = numarg + 1
 if ~ arg10 eq rsi
                mov                             rsi, arg10
 end if
end if
if ~ arg9 eq
 numarg = numarg + 1
 if ~ arg9 eq r14
                mov                             r14, arg9
 end if
end if
if ~ arg8 eq
 numarg = numarg + 1
 if ~ arg8 eq r13
                mov                             r13, arg8
 end if
end if
if ~ arg7 eq
 numarg = numarg + 1
 if ~ arg7 eq r12
```

```
                mov                             r12, arg7
 end if
end if
if ~ arg6 eq
 numarg = numarg + 1
 if ~ arg6 eq r11
                mov                             r11, arg6
 end if
end if
if ~ arg5 eq
 numarg = numarg + 1
 if ~ arg5 eq r10
                mov                             r10, arg5
 end if
end if
if ~ arg4 eq
 numarg = numarg + 1
 if ~ arg4 eq r9
                mov                             r9, arg4
 end if
end if
if ~ arg3 eq
 numarg = numarg + 1
 if ~ arg3 eq r8
                mov                             r8, arg3
 end if
end if
if ~ arg2 eq
 numarg = numarg + 1
 if ~ arg2 eq rdx
                mov                             rdx, arg2
 end if
end if
if ~ arg1 eq
 numarg = numarg + 1
 if ~ arg1 eq rcx
  if ~ arg1 in <ConsoleInHandle,ConIn,ConsoleOutHandle,ConOut,St
                mov                             rcx, arg1
  end if
 end if
end if
                xor                             rax, rax
                mov                             al, numarg
if interface in <ConsoleInHandle,ConIn,ConsoleOutHandle,ConOut,S
                mov                             rbx, [efi_ptr]
                mov                             rbx, [rbx + EFI_SYSTEM_T
else
 if ~ interface eq rbx
                mov                             rbx, interface
```

```
 end if
end if
if arg1 in <ConsoleInHandle,ConIn,ConsoleOutHandle,ConOut,Standa
                mov                     rcx, rbx
end if
if defined SIMPLE_INPUT_INTERFACE.#function
                mov                     rbx, [rbx + SIMPLE_INPUT
else
 if defined SIMPLE_TEXT_OUTPUT_INTERFACE.#function
                mov                     rbx, [rbx + SIMPLE_TEXT_
 else
  if defined EFI_BOOT_SERVICES_TABLE.#function
                mov                     rbx, [rbx + EFI_BOOT_SER
  else
   if defined EFI_RUNTIME_SERVICES_TABLE.#function
                mov                     rbx, [rbx + EFI_RUNTIME_
   else
    if defined EFI_GRAPHICS_OUTPUT_PROTOCOL.#function
                mov                     rbx, [rbx + EFI_GRAPHICS
    else
     if defined EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE.#function
                mov                     rbx, [rbx + EFI_GRAPHICS
     else
                mov                     rbx, [rbx + function]
     end if
    end if
   end if
  end if
 end if
end if
                call                    uefifunc
}

;--------------------------------------------------------------
;*                        Library functions
;--------------------------------------------------------------

section '.text' code executable readable

uefifunc:       ;save stack pointer
                mov                     qword [uefi_rsptmp], rsp
                ;set up new aligned stack
                and                     esp, 0FFFFFFF0h
                ;alignment check on arguments
                bt                      eax, 0
                jnc                     @f
                push                    rax
                ;arguments
@@:             cmp                     al, 11
```

```
                jb                      @f
                push                    rdi
@@:             cmp                     al, 10
                jb                      @f
                push                    rsi
@@:             cmp                     al, 9
                jb                      @f
                push                    r14
@@:             cmp                     al, 8
                jb                      @f
                push                    r13
@@:             cmp                     al, 7
                jb                      @f
                push                    r12
@@:             cmp                     al, 6
                jb                      @f
                push                    r11
@@:             cmp                     al, 5
                jb                      @f
                push                    r10
@@:             ;space for
                ;r9
                ;r8
                ;rdx
                ;rcx
                sub                     rsp, 4*8
                ;call function
                call                    rbx
                ;restore old stack
                mov                     rsp, qword [uefi_rsptmp]
                ret

section '.data' data readable writeable
efi_handler:    dq                      0
efi_ptr:        dq                      0
uefi_rsptmp:    dq                      0
```