**UEFI Memory V E820 Memory**

I use the [EFI STUB](http://zurlinux.com/?p=722) method to boot Fedora 17 directly from a [UEFI](http://blog.fpmurphy.com/2012/08/www.uefi.org) shell without using GRUB. When I got a new Lenovo T430 laptop in July, I found that when I installed a UEFI shell and tried to boot Fedora 17, it simply hung and I had to power cycle the laptop. The problem turned out to be due to the way memory is reported to the Linux kernel by the laptop firmware.

Some background information first before I provide the solution. One of the most vital pieces of information that an operating system (OS) needs in order to initialize itself is a map of available RAM. The only practical way an OS could get that information up to recently on a PC was via the BIOS. [E820](http://http:/en.wikipedia.org/wiki/E820) is a nickname that refers to the facility by which the BIOS of an x86-based computer system reports the memory map to an operating system or a boot loader.

The nickname comes from the fact that it is accessed via the BIOS INT 15h call, by setting the AX register to the value OxE820 (Query System Address Map function.) This function has been available on all PCs since 2002 and was available on most PCs for a long time before that. It reports which memory address ranges (both RAM and ROM) are usable and which are reserved for use only by the BIOS. E820 memory (see *arch/x86/kernel/e820.c*) is one of the first things looked for by a booting Linux kernel (see *../arch/i386/boot/setup.S*.)

The idea of using E820 to query platform physical memory was fine while BIOS was the predominant platform firmware. With the arrival of UEFI firmware, things changed. A UEFI-based OS typically gets the platform physical memory map using the UEFI *GetMemoryMap* interface. This did not sit easily with many Linux kernel developers. According to Linus Torvalds in an email dated July 25th, 2006:

On Mon, 24 Jul 2006, Andrew Morton wrote:  
>  
> > This Patch add an efi e820 memory mapping.  
> >  
>  
> Why?

EFI is this other Intel brain-damage (the first one being ACPI). It’s totally different from a normal BIOS, and was brought on by ia64, which never had a BIOS, of course.

Sadly, Apple bought into the whole “BIOS bad, EFI good” hype, so we now have x86 machines with EFI as the native boot protocol.

The original EFI code in the kernel basically duplicates all the BIOS interfaces (ie everything that looks at a memory map comes in two varieties: the normal and tested BIOS e820 variety, and the usually broken and hacked-up EFI memory map variety).

Translating the EFI memory map to e820 is very much the sane thing to do, and should have been done by ia64 in the first place. Sadly, EFI people (a) think that their stinking mess is better than a BIOS and (b) are historically ia64-only, so they didn’t do that, but went the “we’ll just duplicate everything using our inferior EFI interfaces” way.

Despite E820 being BIOS-specific, the Linux kernel continues to use the E820 map for the in-kernel representation of the memory address map under EFI as well. This caused (and seems to still cause) much pain and angst amongst kernel developers. For example, in a proposal in 2010 by Bill Richardson to “allow empty e820 map if EFI map will be provided later”, H Peter Anvin’s response was:

o! Bloody \*\*\*\* hell no!

This is yet another attempt at doing more of the wrong thing, which not  
only will make it harder to push things that should be earlier in the  
kernel there.

This was settled in 2007 — it is the boot loaders duty to provide a  
memory map. The fact that we allowed a hack in to let the kernel itself  
add additional ranges from EFI has proven to be an utter mistake, and  
this is yet another example of it.

Vetoed in the extreme.

-hpa

The solution (hack), arrived at by the Linux kernel developers in 2009 after a number of false starts was to add a kernel command line option, *add\_efi\_memmap* – to tell the kernel to look at the EFI memory map and use it to fix up various entries in the E820 memory map.

From *…/arch/xx86/platform/efi/efi.c*:

static void \_\_init do\_add\_efi\_memmap(void)

{

void \*p;

for (p = memmap.map; p < memmap.map\_end; p += memmap.desc\_size) {

efi\_memory\_desc\_t \*md = p;

unsigned long long start = md->phys\_addr;

unsigned long long size = md->num\_pages << EFI\_PAGE\_SHIFT;

int e820\_type;

switch (md->type) {

case EFI\_LOADER\_CODE:

case EFI\_LOADER\_DATA:

case EFI\_BOOT\_SERVICES\_CODE:

case EFI\_BOOT\_SERVICES\_DATA:

case EFI\_CONVENTIONAL\_MEMORY:

if (md->attribute & EFI\_MEMORY\_WB)

e820\_type = E820\_RAM;

else

e820\_type = E820\_RESERVED;

break;

case EFI\_ACPI\_RECLAIM\_MEMORY:

e820\_type = E820\_ACPI;

break;

case EFI\_ACPI\_MEMORY\_NVS:

e820\_type = E820\_NVS;

break;

case EFI\_UNUSABLE\_MEMORY:

e820\_type = E820\_UNUSABLE;

break;

default:

/\*

\* EFI\_RESERVED\_TYPE EFI\_RUNTIME\_SERVICES\_CODE

\* EFI\_RUNTIME\_SERVICES\_DATA EFI\_MEMORY\_MAPPED\_IO

\* EFI\_MEMORY\_MAPPED\_IO\_PORT\_SPACE EFI\_PAL\_CODE

\*/

e820\_type = E820\_RESERVED;

break;

}

e820\_add\_region(start, size, e820\_type);

}

sanitize\_e820\_map(e820.map, ARRAY\_SIZE(e820.map), &e820.nr\_map);

}

Whereas E820 was limited to 128 (*E820MAX*) memory regions due to the constrained space in the zeropage, EFI has no such limit. The *do\_add\_efi\_memmap* function adds the memory regions reported by EFI to the kernel’s E820 map, modifies some entries and then sorts and removes overlaps and duplicates. See *…/arch/x86/kernel/e820.c* for more information.

Here is the source code for a simple UEFI command line utility to print out the the EFI memory regions:

|  |  |
| --- | --- |
| 001  002  003  004  005  006  007  008  009  010  011  012  013  014  015  016  017  018  019  020  021  022  023  024  025  026  027  028  029  030  031  032  033  034  035  036  037  038  039  040  041  042  043  044  045  046  047  048  049  050  051  052  053  054  055  056  057  058  059  060  061  062  063  064  065  066  067  068  069  070  071  072  073  074  075  076  077  078  079  080  081  082  083  084  085  086  087  088  089  090  091  092  093  094  095  096  097  098  099  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114 | //  //   Copyright (c) 2012  Finnbarr P. Murphy   All rights reserved.  //  //   License: BSD  //    #include <efi.h>  #include <efilib.h>    #define PAGE\_SIZE 4096    const CHAR16 \*memory\_types[] = {      L"EfiReservedMemoryType",      L"EfiLoaderCode",      L"EfiLoaderData",      L"EfiBootServicesCode",      L"EfiBootServicesData",      L"EfiRuntimeServicesCode",      L"EfiRuntimeServicesData",      L"EfiConventionalMemory",      L"EfiUnusableMemory",      L"EfiACPIReclaimMemory",      L"EfiACPIMemoryNVS",      L"EfiMemoryMappedIO",      L"EfiMemoryMappedIOPortSpace",      L"EfiPalCode",  };      const CHAR16 \*  memory\_type\_to\_str(UINT32 type)  {      if (type > sizeof(memory\_types)/sizeof(CHAR16 \*))          return L"Unknown";        return memory\_types[type];  }      EFI\_STATUS  memory\_map(EFI\_MEMORY\_DESCRIPTOR \*\*map\_buf, UINTN \*map\_size,             UINTN \*map\_key, UINTN \*desc\_size, UINT32 \*desc\_version)  {      EFI\_STATUS err = EFI\_SUCCESS;        \*map\_size = sizeof(\*\*map\_buf) \* 31;    get\_map:      \*map\_size += sizeof(\*\*map\_buf);        err = uefi\_call\_wrapper(BS->AllocatePool, 3, EfiLoaderData, \*map\_size, (void \*\*)map\_buf);      if (err != EFI\_SUCCESS) {          Print(L"ERROR: Failed to allocate pool for memory map");          return err;      }        err = uefi\_call\_wrapper(BS->GetMemoryMap, 5, map\_size, \*map\_buf, map\_key, desc\_size, desc\_version);      if (err != EFI\_SUCCESS) {          if (err == EFI\_BUFFER\_TOO\_SMALL) {              uefi\_call\_wrapper(BS->FreePool, 1, (void \*)\*map\_buf);              goto get\_map;          }          Print(L"ERROR: Failed to get memory map");      }      return err;  }      EFI\_STATUS  print\_memory\_map(void)  {      EFI\_MEMORY\_DESCRIPTOR \*buf;      UINTN desc\_size;      UINT32 desc\_version;      UINTN size, map\_key, mapping\_size;      EFI\_MEMORY\_DESCRIPTOR \*desc;      EFI\_STATUS err = EFI\_SUCCESS;      int i = 0;        err = memory\_map(&buf, &size, &map\_key, &desc\_size, &desc\_version);      if (err != EFI\_SUCCESS)          return err;        Print(L"Memory Map Size: %d\n", size);      Print(L"Map Key: %d\n", map\_key);      Print(L"Descriptor Version: %d\n", desc\_version);      Print(L"Descriptor Size: %d\n\n", desc\_size);        desc = buf;      while ((void \*)desc < (void \*)buf + size) {          mapping\_size = desc->NumberOfPages \* PAGE\_SIZE;            Print(L"[#%02d] Type: %s  Attr: 0x%x\n", i, memory\_type\_to\_str(desc->Type), desc->Attribute);          Print(L"      Phys: %016llx-%016llx\n", desc->PhysicalStart, desc->PhysicalStart + mapping\_size);          Print(L"      Virt: %016llx-%016llx\n\n", desc->VirtualStart, desc->VirtualStart + mapping\_size);            desc = (void \*)desc + desc\_size;          i++;      }        uefi\_call\_wrapper(BS->FreePool, 1, buf);      return err;  }      EFI\_STATUS  efi\_main(EFI\_HANDLE image, EFI\_SYSTEM\_TABLE \*systab)  {      InitializeLib(image, systab);        print\_memory\_map();        return EFI\_SUCCESS;  } |

You need the *gnu\_efi* package installed in order to be able to compile the code. I have placed the above source code and a Makefile on [GitHub](http://github.com/fpmurphy/UEFI-Utilities).

Here is what this utility outputted on my Lenovo T430:

fs0:>efimemory

Memory Map Size: 2736

Map Key: 31803

Descriptor Version: 1

Descriptor Size: 48

[#00] Type: EfiBootServicesCode Attr: 0xF

Phys: 0000000000000000-0000000000001000

Virt: 0000000000000000-0000000000001000

[#01] Type: EfiConventionalMemory Attr: 0xF

Phys: 0000000000001000-000000000005B000

Virt: 0000000000000000-000000000005A000

[#02] Type: EfiBootServicesData Attr: 0xF

Phys: 000000000005B000-000000000005C000

Virt: 0000000000000000-0000000000001000

[#03] Type: EfiBootServicesCode Attr: 0xF

Phys: 000000000005C000-0000000000087000

Virt: 0000000000000000-000000000002B000

[#04] Type: EfiBootServicesData Attr: 0xF

Phys: 0000000000087000-0000000000088000

Virt: 0000000000000000-0000000000001000

[#05] Type: EfiBootServicesCode Attr: 0xF

Phys: 0000000000088000-0000000000090000

Virt: 0000000000000000-0000000000008000

[#06] Type: EfiReservedMemoryType Attr: 0xF

Phys: 0000000000090000-00000000000A0000

Virt: 0000000000000000-0000000000010000

[#07] Type: EfiBootServicesCode Attr: 0xF

Phys: 0000000000100000-0000000000110000

Virt: 0000000000000000-0000000000010000

[#08] Type: EfiConventionalMemory Attr: 0xF

Phys: 0000000000110000-0000000020000000

Virt: 0000000000000000-000000001FEF0000

[#09] Type: EfiReservedMemoryType Attr: 0xF

Phys: 0000000020000000-0000000020200000

Virt: 0000000000000000-0000000000200000

[#10] Type: EfiConventionalMemory Attr: 0xF

Phys: 0000000020200000-0000000040004000

Virt: 0000000000000000-000000001FE04000

[#11] Type: EfiReservedMemoryType Attr: 0xF

Phys: 0000000040004000-0000000040005000

Virt: 0000000000000000-0000000000001000

[#12] Type: EfiConventionalMemory Attr: 0xF

Phys: 0000000040005000-00000000D0489000

Virt: 0000000000000000-0000000090484000

[#13] Type: EfiBootServicesData Attr: 0xF

Phys: 00000000D0489000-00000000D04A9000

Virt: 0000000000000000-0000000000020000

[#14] Type: EfiConventionalMemory Attr: 0xF

Phys: 00000000D04A9000-00000000D2AA2000

Virt: 0000000000000000-00000000025F9000

[#15] Type: EfiBootServicesData Attr: 0xF

Phys: 00000000D2AA2000-00000000D2AB8000

Virt: 0000000000000000-0000000000016000

[#16] Type: EfiReservedMemoryType Attr: 0xF

Phys: 00000000D2AB8000-00000000D2CBA000

Virt: 0000000000000000-0000000000202000

[#17] Type: EfiBootServicesData Attr: 0xF

Phys: 00000000D2CBA000-00000000D3479000

Virt: 0000000000000000-00000000007BF000

[#18] Type: EfiConventionalMemory Attr: 0xF

Phys: 00000000D3479000-00000000D347D000

Virt: 0000000000000000-0000000000004000

[#19] Type: EfiLoaderData Attr: 0xF

Phys: 00000000D347D000-00000000D348D000

Virt: 0000000000000000-0000000000010000

[#20] Type: EfiConventionalMemory Attr: 0xF

Phys: 00000000D348D000-00000000D3490000

Virt: 0000000000000000-0000000000003000

[#21] Type: EfiLoaderData Attr: 0xF

Phys: 00000000D3490000-00000000D3492000

Virt: 0000000000000000-0000000000002000

[#22] Type: EfiConventionalMemory Attr: 0xF

Phys: 00000000D3492000-00000000D3497000

Virt: 0000000000000000-0000000000005000

[#23] Type: EfiLoaderData Attr: 0xF

Phys: 00000000D3497000-00000000D34A6000

Virt: 0000000000000000-000000000000F000

[#24] Type: EfiConventionalMemory Attr: 0xF

Phys: 00000000D34A6000-00000000D398C000

Virt: 0000000000000000-00000000004E6000

[#25] Type: EfiLoaderCode Attr: 0xF

Phys: 00000000D398C000-00000000D3A5A000

Virt: 0000000000000000-00000000000CE000

[#26] Type: EfiConventionalMemory Attr: 0xF

Phys: 00000000D3A5A000-00000000D4A76000

Virt: 0000000000000000-000000000101C000

[#27] Type: EfiBootServicesData Attr: 0xF

Phys: 00000000D4A76000-00000000D55E4000

Virt: 0000000000000000-0000000000B6E000

[#28] Type: EfiConventionalMemory Attr: 0xF

Phys: 00000000D55E4000-00000000D570F000

Virt: 0000000000000000-000000000012B000

[#29] Type: EfiBootServicesData Attr: 0xF

Phys: 00000000D570F000-00000000D5C8F000

Virt: 0000000000000000-0000000000580000

[#30] Type: EfiConventionalMemory Attr: 0xF

Phys: 00000000D5C8F000-00000000D5DBF000

Virt: 0000000000000000-0000000000130000

[#31] Type: EfiBootServicesData Attr: 0xF

Phys: 00000000D5DBF000-00000000D617E000

Virt: 0000000000000000-00000000003BF000

[#32] Type: EfiConventionalMemory Attr: 0xF

Phys: 00000000D617E000-00000000D617F000

Virt: 0000000000000000-0000000000001000

[#33] Type: EfiBootServicesData Attr: 0xF

Phys: 00000000D617F000-00000000D6180000

Virt: 0000000000000000-0000000000001000

[#34] Type: EfiConventionalMemory Attr: 0xF

Phys: 00000000D6180000-00000000D6188000

Virt: 0000000000000000-0000000000008000

[#35] Type: EfiBootServicesData Attr: 0xF

Phys: 00000000D6188000-00000000D7A5A000

Virt: 0000000000000000-00000000018D2000

[#36] Type: EfiConventionalMemory Attr: 0xF

Phys: 00000000D7A5A000-00000000D7C39000

Virt: 0000000000000000-00000000001DF000

[#37] Type: EfiBootServicesCode Attr: 0xF

Phys: 00000000D7C39000-00000000D845A000

Virt: 0000000000000000-0000000000821000

[#38] Type: EfiRuntimeServicesCode Attr: 0xF

Phys: 00000000D845A000-00000000D8505000

Virt: 0000000000000000-00000000000AB000

[#39] Type: EfiRuntimeServicesCode Attr: 0xF

Phys: 00000000D8505000-00000000D865A000

Virt: 0000000000000000-0000000000155000

[#40] Type: EfiRuntimeServicesData Attr: 0xF

Phys: 00000000D865A000-00000000D8BEB000

Virt: 0000000000000000-0000000000591000

[#41] Type: EfiRuntimeServicesData Attr: 0xF

Phys: 00000000D8BEB000-00000000DA65A000

Virt: 0000000000000000-0000000001A6F000

[#42] Type: EfiReservedMemoryType Attr: 0xF

Phys: 00000000DA65A000-00000000DAE35000

Virt: 0000000000000000-00000000007DB000

[#43] Type: EfiReservedMemoryType Attr: 0xF

Phys: 00000000DAE35000-00000000DAE9B000

Virt: 0000000000000000-0000000000066000

[#44] Type: EfiReservedMemoryType Attr: 0xF

Phys: 00000000DAE9B000-00000000DAE9D000

Virt: 0000000000000000-0000000000002000

[#45] Type: EfiReservedMemoryType Attr: 0xF

Phys: 00000000DAE9D000-00000000DAE9F000

Virt: 0000000000000000-0000000000002000

[#46] Type: EfiACPIMemoryNVS Attr: 0xF

Phys: 00000000DAE9F000-00000000DAEF6000

Virt: 0000000000000000-0000000000057000

[#47] Type: EfiACPIMemoryNVS Attr: 0xF

Phys: 00000000DAEF6000-00000000DAF9F000

Virt: 0000000000000000-00000000000A9000

[#48] Type: EfiACPIReclaimMemory Attr: 0xF

Phys: 00000000DAF9F000-00000000DAFDB000

Virt: 0000000000000000-000000000003C000

[#49] Type: EfiACPIReclaimMemory Attr: 0xF

Phys: 00000000DAFDB000-00000000DAFFF000

Virt: 0000000000000000-0000000000024000

[#50] Type: EfiBootServicesData Attr: 0xF

Phys: 00000000DAFFF000-00000000DB000000

Virt: 0000000000000000-0000000000001000

[#51] Type: EfiConventionalMemory Attr: 0xF

Phys: 0000000100000000-000000021E600000

Virt: 0000000000000000-000000011E600000

[#52] Type: EfiReservedMemoryType Attr: 0x0

Phys: 00000000000A0000-00000000000C0000

Virt: 0000000000000000-0000000000020000

[#53] Type: EfiReservedMemoryType Attr: 0x0

Phys: 00000000DB000000-00000000DFA00000

Virt: 0000000000000000-0000000004A00000

[#54] Type: EfiMemoryMappedIO Attr: 0x1

Phys: 00000000F80F8000-00000000F80F9000

Virt: 0000000000000000-0000000000001000

[#55] Type: EfiMemoryMappedIO Attr: 0x1

Phys: 00000000FED1C000-00000000FED20000

Virt: 0000000000000000-0000000000004000

[#56] Type: EfiReservedMemoryType Attr: 0x0

Phys: 000000021E600000-000000021E800000

Virt: 0000000000000000-0000000000200000

fs0>

So back to my original problem. Why I was encountering this problem now for the first time and what was the solution or workaround? After all I have been using UEFI booting for a number of years and have been using the EFI STUB method of booting the kernel since it became available in the Linux kernel. It would appear that all the UEFI firmware I previously used had a CSM (Compatibility Support Module) that could have provided a suitable E820-style memory map to the kernel. Furthermore, until recently I was using either GRUB2 or Red Hat’s EFI-aware GRUB Legacy which both seem to provide an E820-style memory map to the kernel (however I have not checked the source code to ascertain.)

Once I understood the problem, the solution was simple – provide the additional kernel command line option, *add\_efi\_memmap*, on the kernel command line.

add\_efi\_memmap - Include EFI memory map of available physical RAM

If the EFI memory map has additional entries not in the E820 map,

include those entries in the kernels memory map of available physical RAM

Here is how I created a suitable UEFI *BootXXXX* variable containing this kernel option using *efibootmgr*:

# echo " root=UUID=5716419d-0fe4-47a3-bed5-83aeb1c80a5a rd.md=0 rd.lvm=0 rd.dm=0 " \

" KEYTABLE=us SYSFONT=True rd.luks=0 ro LANG=en\_US.UTF-8 add\_efi\_memmap " \

" rhgb quiet initrd=\initramfs.img" | iconv -f ascii -t ucs2 |

efibootmgr --create --gpt --label "Fedora 17 (EFISTUB)" --loader vmlinuz.efi --disk /dev/sdb --append-binary-args -

In case you are wondering after looking at the above, I simply copy the latest *initramfs-\** and *vmlinux-\** to */boot/efi/initramfs* and */boot/efi/vmlinux.efi* respectively. I find this easier to do rather than delete and recreate the BootXXXX variable. I do not bother placing my Fedora kernel in a subdirectory such as *\EFI\FEDORA\* in the ESP (EFI System Partition.) Furthermore the kernel options must be encoded as [UCS2](http://en.wikipedia.org/wiki/UTF-16) instead of ASCII or UTF8 because that is what the UEFI specification calls for.

Here is the output from *efibootmgr -v* after adding the new boot variable.

# efibootmgr -v

BootCurrent: 001A

Timeout: 0 seconds

BootOrder: 001A,0019,0018,000A,0000,0001,0002,0003,0007,0008,0009,000B,000C,000D,000E,000F,0011,0010,0012

Boot0000 Setup

Boot0001 Boot Menu

Boot0002 Diagnostic Splash Screen

Boot0003 Lenovo Diagnostics

Boot0004 Startup Interrupt Menu

Boot0005 ME Configuration Menu

Boot0006 Rescue and Recovery

Boot0007\* USB CD 030a2400d23878bc820f604d8316c068ee79d25b86701296aa5a7848b66cd49dd3ba6a55

Boot0008\* USB FDD 030a2400d23878bc820f604d8316c068ee79d25b6ff015a28830b543a8b8641009461e49

Boot0009\* ATAPI CD0 030a2500d23878bc820f604d8316c068ee79d25baea2090adfde214e8b3a5e471856a35401

Boot000A\* ATA HDD0 030a2500d23878bc820f604d8316c068ee79d25b91af625956449f41a7b91f4f892ab0f600

Boot000B\* ATA HDD1 030a2500d23878bc820f604d8316c068ee79d25b91af625956449f41a7b91f4f892ab0f601

Boot000C\* ATA HDD2 030a2500d23878bc820f604d8316c068ee79d25b91af625956449f41a7b91f4f892ab0f602

Boot000D\* USB HDD 030a2400d23878bc820f604d8316c068ee79d25b33e821aaaf33bc4789bd419f88c50803

Boot000E\* PCI LAN 030a2400d23878bc820f604d8316c068ee79d25b78a84aaf2b2afc4ea79cf5cc8f3d3803

Boot000F\* ATAPI CD1 030a2500d23878bc820f604d8316c068ee79d25baea2090adfde214e8b3a5e471856a35404

Boot0010 Other CD 030a2500d23878bc820f604d8316c068ee79d25baea2090adfde214e8b3a5e471856a35406

Boot0011\* ATA HDD3 030a2500d23878bc820f604d8316c068ee79d25b91af625956449f41a7b91f4f892ab0f604

Boot0012 Other HDD 030a2500d23878bc820f604d8316c068ee79d25b91af625956449f41a7b91f4f892ab0f606

Boot0013\* IDER BOOT CDROM ACPI(a0341d0,0)PCI(16,2)ATAPI(0,1,0)

Boot0014\* IDER BOOT Floppy ACPI(a0341d0,0)PCI(16,2)ATAPI(0,0,0)

Boot0015\* ATA HDD 030a2400d23878bc820f604d8316c068ee79d25b91af625956449f41a7b91f4f892ab0f6

Boot0016\* ATAPI CD: 030a2400d23878bc820f604d8316c068ee79d25baea2090adfde214e8b3a5e471856a354

Boot0017\* PCI LAN 030a2400d23878bc820f604d8316c068ee79d25b78a84aaf2b2afc4ea79cf5cc8f3d3803

Boot0018\* Fedora HD(1,1000,200000,9e81061f-b31f-4a4b-993c-23047ef6ac57)File(\EFI\redhat\grub.efi)

Boot0019\* UEFI Shell HD(1,1000,200000,9e81061f-b31f-4a4b-993c-23047ef6ac57)File(\EFI\fpmurphy\shell.efi)

Boot001A\* Fedora 17 (EFISTUB) HD(1,1000,200000,9e81061f-b31f-4a4b-993c-23047ef6ac57)File(vmlinuz.efi) .r.o.o.t.=.U.U.I.D.=.5.7.1.6.4.1.9.d.-.0.f.e.4.-.4.7.a.3.-.b.e.d.5.-.8.3.a.e.b.1.c.8.0.a.5.a. .r.d...m.d.=.0. .r.d...l.v.m.=.0. .r.d...d.m.=.0. . .K.E.Y.T.A.B.L.E.=.u.s. .S.Y.S.F.O.N.T.=.T.r.u.e. .r.d...l.u.k.s.=.0. .r.o. .L.A.N.G.=.e.n.\_.U.S...U.T.F.-.8. .a.d.d.\_.e.f.i.\_.m.e.m.m.a.p. .r.h.g.b. .q.u.i.e.t. .i.n.i.t.r.d.=.\.i.n.i.t.r.a.m.f.s...i.m.g...

You can see lots of useful information about both the E820 and EFI memory regions using the *dmesg* utility. Here is the relevant output for Fedora 17 on my T430 laptop:

[ 0.000000] Initializing cgroup subsys cpuset

[ 0.000000] Initializing cgroup subsys cpu

[ 0.000000] Linux version 3.5.4-2.fc17.x86\_64 (mockbuild@) (gcc version 4.7.2 20120921 (Red Hat 4.7.2-2) (GCC) ) #1 SMP Wed Sep 26 21:58:50 UTC 2012

[ 0.000000] Command line: root=UUID=48c9b731-d284-436d-b152-39a62240813d rd.md=0 rd.lvm=0 rd.dm=0 KEYTABLE=us SYSFONT=True rd.luks=0 ro LANG=en\_US.UTF-8 add\_efi\_memmap rhgb quiet initrd=\initramfs.img

[ 0.000000] e820: BIOS-provided physical RAM map:

[ 0.000000] BIOS-e820: [mem 0x0000000000000000-0x000000000008ffff] usable

[ 0.000000] BIOS-e820: [mem 0x0000000000090000-0x00000000000bffff] reserved

[ 0.000000] BIOS-e820: [mem 0x0000000000100000-0x000000001fffffff] usable

[ 0.000000] BIOS-e820: [mem 0x0000000020000000-0x00000000201fffff] reserved

[ 0.000000] BIOS-e820: [mem 0x0000000020200000-0x0000000040003fff] usable

[ 0.000000] BIOS-e820: [mem 0x0000000040004000-0x0000000040004fff] reserved

[ 0.000000] BIOS-e820: [mem 0x0000000040005000-0x00000000d1067fff] usable

[ 0.000000] BIOS-e820: [mem 0x00000000d1068000-0x00000000d1269fff] reserved

[ 0.000000] BIOS-e820: [mem 0x00000000d126a000-0x00000000d6a09fff] usable

[ 0.000000] BIOS-e820: [mem 0x00000000d6a0a000-0x00000000dae9efff] reserved

[ 0.000000] BIOS-e820: [mem 0x00000000dae9f000-0x00000000daf9efff] ACPI NVS

[ 0.000000] BIOS-e820: [mem 0x00000000daf9f000-0x00000000daffefff] ACPI data

[ 0.000000] BIOS-e820: [mem 0x00000000dafff000-0x00000000daffffff] usable

[ 0.000000] BIOS-e820: [mem 0x00000000db000000-0x00000000df9fffff] reserved

[ 0.000000] BIOS-e820: [mem 0x00000000f80f8000-0x00000000f80f8fff] reserved

[ 0.000000] BIOS-e820: [mem 0x00000000fed1c000-0x00000000fed1ffff] reserved

[ 0.000000] BIOS-e820: [mem 0x0000000100000000-0x000000021e5fffff] usable

[ 0.000000] BIOS-e820: [mem 0x000000021e600000-0x000000021e7fffff] reserved

[ 0.000000] NX (Execute Disable) protection: active

[ 0.000000] efi: EFI v2.31 by Lenovo

[ 0.000000] efi: ACPI 2.0=0xdaffe014 ACPI=0xdaffe000 SMBIOS=0xdae9e000

[ 0.000000] efi: mem00: type=3, attr=0xf, range=[0x0000000000000000-0x0000000000001000) (0MB)

[ 0.000000] efi: mem01: type=2, attr=0xf, range=[0x0000000000001000-0x0000000000008000) (0MB)

[ 0.000000] efi: mem02: type=7, attr=0xf, range=[0x0000000000008000-0x000000000005a000) (0MB)

[ 0.000000] efi: mem03: type=4, attr=0xf, range=[0x000000000005a000-0x000000000005c000) (0MB)

[ 0.000000] efi: mem04: type=3, attr=0xf, range=[0x000000000005c000-0x0000000000087000) (0MB)

[ 0.000000] efi: mem05: type=4, attr=0xf, range=[0x0000000000087000-0x0000000000088000) (0MB)

[ 0.000000] efi: mem06: type=3, attr=0xf, range=[0x0000000000088000-0x0000000000090000) (0MB)

[ 0.000000] efi: mem07: type=0, attr=0xf, range=[0x0000000000090000-0x00000000000a0000) (0MB)

[ 0.000000] efi: mem08: type=3, attr=0xf, range=[0x0000000000100000-0x0000000000110000) (0MB)

[ 0.000000] efi: mem09: type=7, attr=0xf, range=[0x0000000000110000-0x0000000001000000) (14MB)

[ 0.000000] efi: mem10: type=2, attr=0xf, range=[0x0000000001000000-0x00000000023bb000) (19MB)

[ 0.000000] efi: mem11: type=7, attr=0xf, range=[0x00000000023bb000-0x0000000020000000) (476MB)

[ 0.000000] efi: mem12: type=0, attr=0xf, range=[0x0000000020000000-0x0000000020200000) (2MB)

[ 0.000000] efi: mem13: type=7, attr=0xf, range=[0x0000000020200000-0x0000000040004000) (510MB)

[ 0.000000] efi: mem14: type=0, attr=0xf, range=[0x0000000040004000-0x0000000040005000) (0MB)

[ 0.000000] efi: mem15: type=7, attr=0xf, range=[0x0000000040005000-0x000000007efcf000) (1007MB)

[ 0.000000] efi: mem16: type=2, attr=0xf, range=[0x000000007efcf000-0x0000000080000000) (16MB)

[ 0.000000] efi: mem17: type=7, attr=0xf, range=[0x0000000080000000-0x00000000cea39000) (1258MB)

[ 0.000000] efi: mem18: type=4, attr=0xf, range=[0x00000000cea39000-0x00000000cea59000) (0MB)

[ 0.000000] efi: mem19: type=7, attr=0xf, range=[0x00000000cea59000-0x00000000d1052000) (37MB)

[ 0.000000] efi: mem20: type=4, attr=0xf, range=[0x00000000d1052000-0x00000000d1068000) (0MB)

[ 0.000000] efi: mem21: type=0, attr=0xf, range=[0x00000000d1068000-0x00000000d126a000) (2MB)

[ 0.000000] efi: mem22: type=4, attr=0xf, range=[0x00000000d126a000-0x00000000d1a29000) (7MB)

[ 0.000000] efi: mem23: type=7, attr=0xf, range=[0x00000000d1a29000-0x00000000d1a55000) (0MB)

[ 0.000000] efi: mem24: type=2, attr=0xf, range=[0x00000000d1a55000-0x00000000d1a56000) (0MB)

[ 0.000000] efi: mem25: type=7, attr=0xf, range=[0x00000000d1a56000-0x00000000d1b7a000) (1MB)

[ 0.000000] efi: mem26: type=1, attr=0xf, range=[0x00000000d1b7a000-0x00000000d200a000) (4MB)

[ 0.000000] efi: mem27: type=7, attr=0xf, range=[0x00000000d200a000-0x00000000d2f27000) (15MB)

[ 0.000000] efi: mem28: type=4, attr=0xf, range=[0x00000000d2f27000-0x00000000d34d5000) (5MB)

[ 0.000000] efi: mem29: type=7, attr=0xf, range=[0x00000000d34d5000-0x00000000d34ef000) (0MB)

[ 0.000000] efi: mem30: type=4, attr=0xf, range=[0x00000000d34ef000-0x00000000d46b8000) (17MB)

[ 0.000000] efi: mem31: type=7, attr=0xf, range=[0x00000000d46b8000-0x00000000d46ca000) (0MB)

[ 0.000000] efi: mem32: type=4, attr=0xf, range=[0x00000000d46ca000-0x00000000d600a000) (25MB)

[ 0.000000] efi: mem33: type=7, attr=0xf, range=[0x00000000d600a000-0x00000000d618c000) (1MB)

[ 0.000000] efi: mem34: type=3, attr=0xf, range=[0x00000000d618c000-0x00000000d6a0a000) (8MB)

[ 0.000000] efi: mem35: type=5, attr=0x800000000000000f, range=[0x00000000d6a0a000-0x00000000d6ab0000) (0MB)

[ 0.000000] efi: mem36: type=5, attr=0x800000000000000f, range=[0x00000000d6ab0000-0x00000000d6c0a000) (1MB)

[ 0.000000] efi: mem37: type=6, attr=0x800000000000000f, range=[0x00000000d6c0a000-0x00000000d77b3000) (11MB)

[ 0.000000] efi: mem38: type=6, attr=0x800000000000000f, range=[0x00000000d77b3000-0x00000000da65a000) (46MB)

[ 0.000000] efi: mem39: type=0, attr=0xf, range=[0x00000000da65a000-0x00000000dae35000) (7MB)

[ 0.000000] efi: mem40: type=0, attr=0xf, range=[0x00000000dae35000-0x00000000dae9b000) (0MB)

[ 0.000000] efi: mem41: type=0, attr=0xf, range=[0x00000000dae9b000-0x00000000dae9d000) (0MB)

[ 0.000000] efi: mem42: type=0, attr=0xf, range=[0x00000000dae9d000-0x00000000dae9f000) (0MB)

[ 0.000000] efi: mem43: type=10, attr=0xf, range=[0x00000000dae9f000-0x00000000daef6000) (0MB)

[ 0.000000] efi: mem44: type=10, attr=0xf, range=[0x00000000daef6000-0x00000000daf9f000) (0MB)

[ 0.000000] efi: mem45: type=9, attr=0xf, range=[0x00000000daf9f000-0x00000000dafda000) (0MB)

[ 0.000000] efi: mem46: type=9, attr=0xf, range=[0x00000000dafda000-0x00000000dafff000) (0MB)

[ 0.000000] efi: mem47: type=4, attr=0xf, range=[0x00000000dafff000-0x00000000db000000) (0MB)

[ 0.000000] efi: mem48: type=7, attr=0xf, range=[0x0000000100000000-0x000000021e600000) (4582MB)

[ 0.000000] efi: mem49: type=0, attr=0x0, range=[0x00000000000a0000-0x00000000000c0000) (0MB)

[ 0.000000] efi: mem50: type=0, attr=0x0, range=[0x00000000db000000-0x00000000dfa00000) (74MB)

[ 0.000000] efi: mem51: type=11, attr=0x8000000000000001, range=[0x00000000f80f8000-0x00000000f80f9000) (0MB)

[ 0.000000] efi: mem52: type=11, attr=0x8000000000000001, range=[0x00000000fed1c000-0x00000000fed20000) (0MB)

[ 0.000000] efi: mem53: type=0, attr=0x0, range=[0x000000021e600000-0x000000021e800000) (2MB)

[ 0.000000] DMI 2.7 present.

[ 0.000000] DMI: LENOVO 2342CTO/2342CTO, BIOS G1ET69WW (2.05 ) 09/12/2012

[ 0.000000] e820: update [mem 0x00000000-0x0000ffff] usable ==> reserved

[ 0.000000] e820: remove [mem 0x000a0000-0x000fffff] usable

[ 0.000000] No AGP bridge found

[ 0.000000] e820: last\_pfn = 0x21e600 max\_arch\_pfn = 0x400000000

[ 0.000000] MTRR default type: uncachable

[ 0.000000] MTRR fixed ranges enabled:

[ 0.000000] 00000-9FFFF write-back

[ 0.000000] A0000-BFFFF uncachable

[ 0.000000] C0000-FFFFF write-protect

[ 0.000000] MTRR variable ranges enabled:

[ 0.000000] 0 base 0FFC00000 mask FFFC00000 write-protect

[ 0.000000] 1 base 000000000 mask F80000000 write-back

[ 0.000000] 2 base 080000000 mask FC0000000 write-back

[ 0.000000] 3 base 0C0000000 mask FE0000000 write-back

[ 0.000000] 4 base 0DC000000 mask FFC000000 uncachable

[ 0.000000] 5 base 0DB000000 mask FFF000000 uncachable

[ 0.000000] 6 base 100000000 mask F00000000 write-back

[ 0.000000] 7 base 200000000 mask FE0000000 write-back

[ 0.000000] 8 base 21F000000 mask FFF000000 uncachable

[ 0.000000] 9 base 21E800000 mask FFF800000 uncachable

[ 0.000000] x86 PAT enabled: cpu 0, old 0x7040600070406, new 0x7010600070106

[ 0.000000] e820: last\_pfn = 0xdb000 max\_arch\_pfn = 0x400000000

[ 0.000000] initial memory mapped: [mem 0x00000000-0x1fffffff]

[ 0.000000] Base memory trampoline at [ffff880000054000] 54000 size 24576

[ 0.000000] init\_memory\_mapping: [mem 0x00000000-0xdaffffff]

[ 0.000000] [mem 0x00000000-0xdaffffff] page 2M

[ 0.000000] kernel direct mapping tables up to 0xdaffffff @ [mem 0x1f923000-0x1fffffff]

[ 0.000000] init\_memory\_mapping: [mem 0x100000000-0x21e5fffff]

Well, by now you shold now have a fairly good understanding of the difference between E820 and EFI memory maps amd how to solve the problem of booting your Linux kernel if your particular UEFI implementation causes a problem in this area.

As an aside, I see no reason to use GRUB, Lilo or any other boot loader when you are on a UEFI-enabled platform. Typically, UEFI comes with its own boot manager that is capable of booting Windows, Linux, BSD and more. Moreover, your boot time is significantly faster when you eliminate the GRUB bootloader.

- See more at: http://blog.fpmurphy.com/2012/08/uefi-memory-v-e820-memory.html#sthash.ZDaZHxCQ.dpuf