A Proposal for a Free EFI
see free-efi.org or
openefi.org
(note: EFI is not trademark anyone, AFAIK)

Ron Minnich

LA-UR-05-7561

What's EFI?

- A BIOS replacement
  - Yes
- An OS?
  - Yes, clearly, if you read the docs
- Both?
  - Yes, a very heavy BIOS
  - Or a new, proprietary (non-open-source) OS
  - Take your pick

My view

- EFI is a proprietary, closed-source, "members only" OS *project* designed to replace the PC BIOS *but* preserve the proprietary nature of the BIOS
- See the uefi.org:
- Design Guides
  - UEFI Design Guides are available for download in the ***Members Only*** area.
- Supplemental Test Tools
  - Additional UEFI test tools are available in the ***Members Only*** area.
- Technical Information
  - Additional UEFI test tools are available in the ***Members Only*** area.

Who are the members?

- * AMD
- * American Megatrends Inc.
- * Dell
- * Hewlett Packard
- * IBM
- * Insyde
- * Intel
- * Microsoft
- * Phoenix Technologies

Who are the members?

- i.e., 3 proprietary BIOS companies, a proprietary software company, and hardware companies
- Interestingly enough, there are *fewer* companies contributing to EFI than to LinuxBIOS
- A driving force for EFI is the desire to allow vendors to maintain proprietary information
  - "Silicon IP"
  - I've been told this by the guys who wrote it

How do you join?

- Pony up your $2500
- You can “adopt” it for free
- You just can't contribute to it
- We've seen this all before
- Anybody remember I2O or IBTA?
- The same closed-member development model
- This type of model is usually a recipe for initial success, and long-term failure (see: I2O)

Problems with the EFI "members-only" development model

- This type of model leads to anything but innovation
- Tends to lead to "consensus of conventional wisdom"
- Linux could never have grown out of this process
- What does grow? What is EFI like?

EFI is an OS that looks much like a 1960s OS

- Ignores all the lessons of the last 35 years
- What are those lessons?
- The Unix lesson: uniform interface, different resources
  - e.g. The kernel interface to drivers
  - Uniform interface to scsi, network, console, graphics, etc.
- What's that look like in EFI?

Some EFI device documents

- Acpi.pdf CpuIo.pdf FAS.pdf Hob.pdf PciPlat.pdf S3Resume.pdf
- AcpiTblStor.pdf Csm.pdf Ffs.pdf HotPlugPci.pdf PeiCis.pdf SmbusHostCont.pdf
- BootScript.pdf DataHub.pdf Fv.pdf IdeCont.pdf PeiCis_0-91.pdf SmbusPpi.pdf
- BootScript_0-91.pdf DataHubSubclass.pdf FvBlock.pdf MemSubclass.pdf PlatIde.pdf SmmCis.pdf
- CacheSubclass.pdf DxeCis.pdf HII_0-91.pdf MiscSubclass.pdf ProcSubclass.pdf StatusCodes.pdf
- Capsule.pdf DxeCis_0-91.pdf Hii.pdf PciHostBridge.pdf Recovery.pdf StatusCodes_0-92.pdf
-

And they're not even that open:

- Pdftotext on some files shows:
- Error: Copying of text from this document is not allowed.
- Great: even the .pdf's have restrictions
- But what is clear: the interfaces are all very different

Let's look at SMBUS interface

- EFI_SMBUS_HC_PROTOCOL.Notify() Summary Allows a device driver to register for a callback when the bus driver detects a state that it needs to propagate to other drivers that are registered for a callback.
- I could not easily paste it (cut to acroread)
  - Page 14 of SmbusHostCont.pdf
- But, goodness, device drivers registering callbacks? What's going on here? Is this a BIOS?
- What do other devices look like?
- They're utterly different

What else does EFI have besides drivers?

- Would you believe a file system?
  - Flat only: no hierarchy.
- How about timers?
- How about protocols?
- How about network drivers?
- How about interrupts?
- Yup, it's an OS all right
- The .pdf's alone are 10 MB or more

Speaking of the file system ...

- “The following FvCheck() pseudo code must be executed during FFS initialization to avoid file system corruption. If at any point a failure condition is reached, then the firmware volume is corrupted and a crisis recovery is initiated.” -- from Ffs.pdf
- No, you don't want to see that pseudo-code
- But at least it has an fsck

Where will all this code come from?

- You, the vendors
- All new, all written from scratch
- Needing to be tested in four different OS environments
  - Since there are >= 4 totally independent EFI implementations
- The burden on vendors is going to be greatly increased

Another problem with the model

- The basic idea is that you create a spec
  - Via uefi.org
- And then people write software conforming to the spec
- Spec is a lot of alphabet soup, "must", "shall", and "should"
- And pseudo-code
- And interface definitions

The “o pen spec” model

- Has this ever worked
- Not nearly as frequently as it has failed
  - Infiniband (success required open source, not spec)
  - ISO/OSI
  - The many years of attempts to unify Unix
- And it has never worked for anything as complex as an OS
- Tends to break down badly with multiple implementations

How many EFI implementations are there?

- Currently, I believe there 4
- This has special meaning to vendors:

“We HATE having two driver bases to deal with” -- vendor, to us

- As in “W indows” and “L inux”
- Ah, well, if you don't like having two, how will you like having *six*?
- Because, fact is, we've already seen differences between EFIs from different vendors
- EFI is going to bring us back to the days of the Unix split, when vendors had as many drivers as Unix implementations
- It was hard then, and it will be hard(er) now

The driver mess

- In Unix, in the later years, drivers:
  - Had to be written for several vendor variants of Unix
  - And for *each* version of *each* vendor Unix
  - #ifdef hell; Makefile hell; not a lot of fun at all
- Vendors: you will get to do this again for EFI
  - And, you won't have the freedom of action you have had with PCI expansion ROMs
- In this case it will be “All pain, no gain”
  - And there really will be no gain

Why is EFI doing this?

- A desire to improve the BIOS situation
  - Break the chains of the legacy BIOS
- Preserve chip IP
  - The big goal: make life safe for binary drivers
- But what do these have to do with writing a new flat file system?
- Or TCP protocol?
- A new proprietary, multiple-implementation OS is a mistake

Is there an alternative

- Yes.
- If you're going to need an OS for your firmware, there's no need to write a new one
- Linux is already there
- It is smaller, less complex, faster, more reliable, and has far more contributors than EFI
- EFI does make a few contributions, however

EFI contributions

- EFI shows that you *can* run a full-up OS as your BIOS
  - Current EFI implementations notwithstanding:
  - It really doesn't have to be slow
  - Makes the case for Linux-as-BIOS even stronger
- EFI may help us get around chipset setup issues
  - We will need the ability to run EFI modules under Linux
- EFI gets us that big fat flash part

EFI contributions

- EFI shows that LinuxBIOS makes sense – an OS should be the BIOS
- EFI will force vendor hardware to provide systems that support Linux-as-BIOS requirements
- Overall, EFI will make the vendor environment friendlier to LinuxBIOS
- While we don't want EFI, EFI will help LinuxBIOS in the long run

How to build a truly open EFI

- No membership fees
- No huge specs – the 'giant spec' always fails
- Start with the assumption that Linux is your base
- Use Linux modules where possible for chipset setup
- Have the ability to run EFI modules as needed
  - Or do a binary translation to Linux modules
- Linux-as-BIOS becomes a superset of EFI!

Where do we go from here

- First, figure out if anyone really cares about EFI
  - I know I don't
  - We've seen no use for it at LANL
  - We've seen very negative results from systems that have it (slow, buggy, hard to set up, etc.)
- OK, maybe nobody wants it, but we might get stuck with it anyway
  - Intel's pushing very hard for it
  - Vendors want to maintain binary drivers

If we are going to have to live with EFI or something like it

- Then build systems up with Linux-as-BIOS, i.e. LinuxBIOS
- Figure out how to run EFI modules in emulation (via user mode is most desirable)
  - We don't need EFI file systems, etc.; just drivers
- Figure out if we can remove and replace EFI modules with open-source Linux drivers
- Develop the ability to take an EFI-based system and turn it into a LinuxBIOS-based system

In other words

- Make it advantageous for vendors to write binary drivers, but for Linux, not for EFI
- Or make it possible to run those EFI binaries in Linux
- Either way, it should be clear to vendors that the path of least resistance is through Linux
  - Avoids need to write 6 drivers

If there is interest in the vendor community

- We are interested in working with them on this problem
- Two approaches
  - Build Linux-as-BIOS, then show vendors they don't need binary EFI modules, they need binary Linux modules
  - Get EFI emulation working under Linux

Summary

- Goals of EFI are understandable
  - Break chains of BIOS
  - Find a way to preserve “silicon IP”
- The path taken – a new, proprietary, “spec-driven” OS – is a mistake
- Vendors will have to at minimum triple the number of drivers they write
- EFI will help in one way – new mainboards will better accommodate LinuxBIOS

Linux-as-BIOS is the better path

- Use Linux as the BIOS, not EFI
  - Leverage huge Linux code, knowledge base
- Use Linux modules/programs for startup, not EFI binary modules
- Be willing to accommodate binary drivers in the Linux-as-BIOS framework
  - Higly undesirable, but acceptable
- We are willing to work with vendors on this idea