Power Management and Linux

In recent years as computers become more power hungry and more people have access to notebook computers, power management has become more important to the wider computer using population. The power management built into Linux allows it to perform fairly well in such benchmarks as battery life and power consumption.

We will discuss the history of the Linux power management implementation, the internal kernel API's involved and the future.

1. Notes

1.1. Original presentation

The original presentation of this talk occurred in room B of the Ottawa Linux Symposium, Ottawa Congress Centre, Ottawa, Ontario, Canada on the 22nd of July, 2000 at 11:30 local time. This presentation was given by Stephen Rothwell.

1.2. Presenter bio

Stephen Rothwell came to Linuxcare from NEC Australia. At NEC he helped set up and administer NEC's external Internet connection and intranet. He also set up, maintained and supported Linux firewalls and Web servers for various NEC customers. Stephen develops and maintains an Advanced Power Management driver for Linux licensed under the GPL. He is a member of the Australian UNIX and Open Systems

Users Group (AUUG) and has presented papers at two of its Canberra Technical Conferences. Stephen was one of the organizers of the Canberra Linux Users group and is also one of the administrative team of the Canberra PCUG's Internet service. Stephen began programming by teaching an IBM 360 how to cheat at Blackjack, and has been known to tell three funny jokes in one year.

1.3. Presentation recording details

This transcript was created using the OLS-supplied recording of the original live presentation. This recording is available from ftp://ftp.linuxsymposium.org/ols2000/2000-07-22_12-58-04_B_64.mp3

The recording has a 64 kb/s bitrate, 32KHz sample rate, mono audio (due to the style of single microphone recording used) and has a file size of 29475648 bytes. The MD5 sum of this file is: 383c433727dead85fa73741adbf7fda2

1.4. Creation of this transcript

1.4.1. Request for corrections

This transcript was not created by a professional transcriptionist; it was created by someone with technical skills and an interest in the presented content. There may be errors found within this transcript; we ask that you report them to using the bug tracking interface described at http://olstrans.sourceforge.net/bugs.php3

1.4.2. Tools used in transcript creation

This transcription was made from the MP3 recording of the original presentation, using XMMS for playback and lyx (with docbook template) for the transcription.

1.4.3. Format of transcript files

The transcribed data should be available in a number of formats so as to provide more ready access to this data to a larger audience. The transcripts will be available in at least HTML, SGML and plain ASCII text formats; other formats may be provided.

1.4.4. Names of people involved with this transcription

This transcript was created by Jacob Moorman of the Marble Horse Free Software Group (whose pages live at http://www.marblehorse.org). He may be reached at roguemtl@marblehorse.org

The primary quality assurance for this document was performed by Stephanie Donovan. She may be reached at sdonovan@achilles.net

1.4.5. Notes related to the use of this document

This document is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. While quality assurance checks on this transcript were performed, it was not created nor checked by a professional transcriptionist; the technical accuracy of this transcript is neither guaranteed nor confirmed. Please refer to the original audio recording of this talk in the event confirmation of the speaker's actual statements are needed.

1.4.6. Ownership of the content within this transcript

1.5. Markup used in this transcript

1.5.1. Time markers

At the end of each paragraph within the body of this transcript, a time offset is listed, corresponding to that point in the MP3 recording of the presentation. This time marker is emphasized (in document formats in which emphasis is supported) and is placed within brackets at the very end of each paragraph. For example, *05m, 30s* states that this paragraph ends at the five-minute, thirty-second mark in the MP3 recording.

1.5.2. Questions and comments from the audience

These recordings were created using a bud microphone attached to the speaker during their presentation. Due to the inherent range limitations of this type of microphone, some of the comments and questions from the audience are unintelligible. In cases where the speaker repeats the audience question, the question shall be omitted and a marker will be left in its place. Events which happen in the audience shall be bracketed, such as: The audience applauds.

Further, in cases where the audience comments or questions are not repeated by the speaker, they shall be included within this transcript and shall be enclosed within double quotes to delineate that the statements come from the audience, not from the speaker.

1.5.3. Editorial notes

The editor of this transcript, the transcriptionist (if you will), and the quality assurance resource who have examined this transcript may each include editorial notes within this transcript. These shall be placed within brackets.

1.5.4. Paragraph breaks

The paragraph breaks within this transcript are very much arbitrary; in many cases they represent pauses or breaks in the speech of the speaker. In other cases, they have been inserted to allow for enhanced clarity in the reading of this transcript.

1.5.5. Speech corrections by the speaker

During the course of the talk, the speaker may correct himself or herself. In these cases, the corrected speech will be placed in parenthesis. The reader of this transcript may usually ignore the parenthised sections as they represent corrected speech. For example: My aunt once had (a dog named Spot, sorry) a cat named Cleopatra.

1.5.6. Unintelligible speech

In sections where the speech of the author or audience has been deemed useful, but unintelligible by the transcriptionist or by the quality assurance resource, a marker will be inserted in their places, unintelligible. Several attempts will be made to correct words and phrases of this nature. In cases where the unintelligible words or phrases are clearly not of importance to the meaning and understanding of the sentence, they may be omitted without marker insertion.

2. Transcript

I’m Stephen Rothwell, I’m the Linux Advanced Power Management maintainer; advanced is a bad word...

There is a comment from an audience member.

Sometimes it works.

More advanced than turning it off. The audience laughs.

Ahh, yeah. Before I start, I should ask: none of you guys actually program BIOSes, do you? Oh good. The audience laughs. *00m, 26s*

This is basically what I'm going to talk about today. A bit of motivation about why we want power management, a bit about why I did it. A bit of history. And then I'm going to talk about APM, which is what I did. Then I'll talk a little about ACPI, which is what I didn't do. And we'll talk about the implementation, how we did it and how you use it. And talk about device APIs and if there are any device driver writers here, I urge you to listen and take note. Down to that point, I'll be talking about Intel and mainly APM. As I keep telling Paul Mackerras, talking about Intel covers like 90 of the market, so the rest are just noise. I'll talk a little bit about other architectures, like PPC and Transmeta. Then I've got some references and I'll take some questions. *01m, 25s*

The first time I did this talk, I thought about motivation and it sort of came down to justification, rather than motivation. But I was thinking about this and justification is easy. How many people here have got a laptop? Okay. How many people traveled more than two hours to get here? That's the justification. The other justification is that these days people build these great big farms of PCs or have PCs all over their corporations, like thousands of them, and they're all sitting there burning power. Especially if they're running Linux, because you don't shut them down every night. And we want to reduce power bills, turn fans off, save noise; things like that. *02m, 13s*

My motivation was the fact that I got a laptop about seven years ago when not so many people had laptops. And it was running Windows. And I thought: this is a bit of a silly thing to do, so I'll put Linux on it. So I put Linux on it, sat down and put it on my lap and almost burnt myself. This wasn't good. The other thing was, of course, that on Windows power management sort of worked. It did things, in particular it would turn itself off when you shut it down. And Linux didn't do that; and I thought anything Windows can do, we've got to be able to do as well. *02m, 54s*

The other motivation is longer battery life. It's like the flight between Sydney Australia and LA Los Angeles, California, United States of America is like fourteen hours. On my first laptop, the battery would last about an hour if I was very lucky. Which doesn't get you very far. And these days, your Pentium III (P3) laptops and some of the Pentium II laptops come with fans which make a bit of noise. If you turn the fan off and you don't do any power management, the P3 is liable to melt through the bottom of your laptop, straight through your knee. And that hurts a lot. One thing you'll notice on most Linux laptops these days is that the fans don't run. Well, they don't on mine, anyway, except when you build kernels, and then they run. *03m, 44s*

A bit of history... Before 1992, there was total chaos, because I hadn't thought about this yet, and neither had anybody else. Linux, of course, existed only just. Linux on laptops, there weren't that many laptops. But somebody was thinking about this; unfortunately, the somebody was Microsoft and Intel. And in 1992 they came up with this thing called Advanced Power Management. It was Advanced because before that, there was none. *04m, 16s*

So, a step forward. Mid-1993, I got my first laptop; things changed. I didn't know about APM at this point and I thought: let's see what this hardware is and what we can do with it. Unfortunately, I couldn't do anything with it because there were no hardware specs. So you're sort of stuck. But I was looking through a book shop one day and I saw a book on the shelf called, Intel's SL Architecture. I thought: interesting. I opened the book and what it talks about is power management and I thought: good, hardware specs. Unfortunately, it was for a 386; I had a 486 laptop and things were a bit different. So I tried to do some of this stuff. *05m, 08s*

The first thing you learn about power management and Intel's way of doing things is what they invented was a new mode on your processor. You all know you've got real mode, you've got protected mode; well they invented this thing called system management mode. When I described this thing to a friend of mine, he got really scared, because system management mode is a mode you can get into and the operating system can't do anything about it. You end up being in 32-bit real mode and all interrupts are disabled, including NMIs. And there's no way of stopping the processor from going into this mode. You don't get back unless the thing that's running puts you back. *05m, 49s*

I thought: okay, this looks fun. I switched my machine into system management mode and it just crashed. The speaker laughs. I never figured out how to do this. It's probably better documented these days. So all of the late 386es, most of the 486es and all of the Pentiums have system management mode. One of the things that it would be good for, if someone really felt like it, would be writing a really good debugger, because you can just stop the operating system dead and you've got access to all of your memory. It would be hard. *06m, 19s*

September 1993, a new APM spec came out. The APM version 1.0 spec left an enormous amount to your imagination, which is never good in a specification. APM 1.1 tried to clarify a lot of that. It sort of did. This is about when I found the APM spec. Unfortunately, I found it on the Microsoft site, but it was on the Intel site as well, so I downloaded it from there. And I started hacking away at it. I got the machine to power off; that wasn't too hard. I also discovered I needed my first workaround for a BIOS bug, almost immediately. *07m, 05s*

In 1994, we put our first APM into the kernel. Things get better, I get lots of bug reports, do lots more stuff. In 1996, we get APM version 1.2, which clarified an enormous amount, which was really good because people were asking all these questions and saying: well the spec doesn't say anything about that, so I'm not going to do it. Unfortunately in 1.2 they clarified all this stuff, so I had to do it to make it work. The speaker laughs. *07m, 32s*

Microsoft and Intel were realizing that APM was really a bad mistake. And it sort of was pretty clear from the beginning. So in December, they released this thing called the ACPI spec. ACPI stands for Advanced Configuration and Power Interface. They like advanced. There were no non-advanced things, only advanced things. *07m, 53s*

Well version 1.2 of the APM spec, which was the latest, is about 60 to 70 pages. The ACPI spec, which covers more than power management, is 300 to 400 pages. It's like this thick. I've never read the whole thing. It's a good way of getting to sleep at night. It's amazing. *08m, 20s*

The ACPI version 2 is in draft, as of January or February of this year. ACPI version 2 is interesting, because you have to actually sign up on their website and join the consortium to get a hold of the spec, so I haven't got a copy, because I don't necessarily believe that people should do that. (Where do we go from here?) *08m, 51s*

In ACPI, Toshiba joined Intel and Microsoft, which probably helped because at least Toshiba manufactured laptops. So they knew the issues here. With version 2, Compaq and Phoenix joined too, which was useful because Phoenix writes BIOSes. It doesn't necessarily mean that they do it ACPI right, though. *09m, 17s*

What is APM? APM is actually a BIOS interface specification. This is why this is a bad mistake; because somebody else is writing the software and we have no control over it. And they tried to define the interface between the operating system and the BIOS, which comes supplied with your laptop or whatever your PC is. As I said, they left a lot of things to your imagination. I don't think Microsoft has a lot of experience in writing real APIs. As Andrew Tridgell would know, you might be okay afterwards, to do what you implemented. *09m, 59s*

This is an example from Miguel de Icaza of why you don't do policy low-level, because the APM basically defines policy in the BIOS. Which basically means you're stuck with whatever the BIOS implementer thought was a good idea, which is not always right. Alternatively, they provide this setup; you get a setup screen when you boot into your BIOS config and you set all of these timeouts and things. Not very useful when you don't want to reboot very often. *10m, 36s*

What APM allows you to do is for the operating system to cooperate with the BIOS, and a bit the other way as well. What can happen is the operating system can inform the BIOS when it considers the machine idle, which is useful. My first laptop, the BIOS was badly written, and what the BIOS considered to be idle was that you had stopped typing, which is not very useful when you're doing kernel compiles. What I discovered is that my kernel compiles went 8-10 times faster if I typed. So I'd start a compile and go away and come back three hours later and find it's got through like a third of it and start typing and zoom noise it would finish. *11m, 20s*

So if the operating system can tell the BIOS when it is idle, this is much better because the operating system actually knows. The operating system actually knows that there are processes out there doing things and can stop lowering the speed of the CPU. The operating system can, in theory, manage the power of devices. That's what the theory says. In practice, it's actually really hard to figure out what the BIOS calls each device. *11m, 53s*

Basically, you can say: turn the power off on this type of device; all of this type of device or this particular device of this type. There's no way to find out what they are. There's no enumeration, which is a problem. And the operating system can manage the power overall. Again the BIOS will do things sort of its own, it will basically tell the operating system: we're going to suspend now because I think we're idle. Which is sometimes good, for instance, because the BIOS knows when the battery is going dead and it can say to the operating system; we're gone. The operating system doesn't get to say anything, but at least you know it happened. *12m, 36s*

Some thing that are a problem here, like only the BIOS has access to the sleep button. So when you push the sleep button on here (which I won't do), the BIOS gets told about it. And what the operating system gets told is that the user did something and would like to suspend. But the operating system can then say: okay, we'll suspend; or no we won't. But it can't decide to do something else, for instance. It's very coarse-grained. Basically, you can suspend, you can resume, you don't get to decide to resume. We need a lot more than that. *13m, 23s*

ACPI... ACPI is actually a lot more than power management. In theory, it's supposed to provide the operating system with a description of all of the hardware that the motherboard at least knows about. Which on a laptop, as you would imagine, is really useful because everything is on the motherboard. In practice, the BIOS writers get in the way again; and they tell you about some of the things, but they don't tell you about everything. They tell you about what they want you to know about. *13m, 59s*

On top of that, the ACPI provides you with methods or functions for doing things to devices, so you don't actually have to know how to do things to a device; you can basically look up in the ACPI in the particular method, like the method to turn a device off, and you can execute it. These methods are written in an interpreted language called AML, which stands for ACPI Machine Language, which is pretty imaginative. *14m, 34s*

The problem with this is that you need the interpreter for AML in order to actually run the methods. The ACPI guys for Linux have actually written such an interpreter for Linux and a few weeks ago it went into the kernel. The big warning in the configuration help says: if you select this, you will add 120K to your kernel; which everybody wants. The speaker laughs. Somebody was saying to me the other day that Microsoft is forcing us all to be like Microsoft, because if you implement ACPI, you end up with all this stuff. *15m, 10s*

The difference with ACPI is that the operating system has complete control. Basically, the ACPI says you collar these devices; you can do these things to them, go for it. This is good, because the operating system actually knows what it wants to do. It's exported the policy at least up one level. The operating system can then export the policy wherever it wants, which is what we do because we don't want to make actual decisions about things; we'll let someone else do that. *15m, 44s*

You've got access to the sleep buttons and things. Basically, if the user pushes the sleep button, the operating system gets an event that says: the user pushed the sleep button, do what you like with it. You could make it pop up nethack. The audience laughs. There's still some dependence on the BIOS writers. *16m, 04s*

Microsoft, in Windows 98, the first edition of Windows 98 depended on ACPI; if it was there, it would use it. This caused all sorts of interesting problems. And the first bug release and certainly the second edition of Windows 98 disabled the use of ACPI for any BIOS written before the first of December, 1999, which gives you some indication of how well they were going. It then had a white list that says: if this is a BIOS I recognize and I know works, then use it; otherwise, we won't. *16m, 37s*

Which is the inverse of what you want to do; maybe you want to say that you know this one is really bad, and all the rest are okay, but that's not the way it worked. So basically, there was incorrect information, methods wouldn't work; there was incomplete information, so you could do some of the stuff, but doing some of the stuff is not enough, because this other device over here doesn't recover when you resume because you didn't suspend it properly. All these sort of things go on. *17m, 07s*

You've got much finer-grained power management. The spec defines four power levels for your CPU, four power levels for each device, and (five power levels) five sleep modes which are effectively for the system as a whole... it's going to be a pain. Basically, you can have the CPU running either at different speeds or whatever you need to do to lower its power usage. The only difference between the different levels is how long it takes you to get back to running full speed. So given this information, you can decide if you want to put your CPU into a low-power mode because it might take you 10ms to get back from power mode C1, whereas it might take you 100ms to get back from C3 and if you need to have some sort of response within 20ms, then you don't want to go to sleep that's going to take 100ms to get back. *18m, 12s*

So you've got all of this information now and you can make rational decisions. The sleep modes, again, adjust how long it takes to get back, how much effort it takes to get back. Obviously the bottom sleep mode is basically power off. Which is way down there. The level above that is effectively hibernation; power off and you can come back again afterwards. They also, just for the heck of it, put in this thing called S4 BIOS, which is BIOS-assisted hibernation. So you can just say to the BIOS: I want to go to sleep now, but save everything. So that's effectively the old APM-style hibernation, which is useful of course. *18m, 54s*

But there's a guy up there who's written a software suspend patch, which you probably all know about. Which is actually useful under Linux because it only saves what we need to save. Like, we don't save all of these pages that have got nothing useful in them. Whereas the hibernation built-in to the BIOS has to save all of memory, all of video state, all of video memory and anything else it can think of into this enormous file. *19m, 22s*

Let's talk a little bit about implementation. What have you got to do? The operating system; first thing it has to do is initialize the APM subsystem and enable it; you've got to do both, I guess that makes sense. This, of course, is the very first thing you have to do. The very first thing I had to do was the very first BIOS bug I found. *19m, 49s*

I had a laptop from NEC because I worked for NEC and it came with a docking station. And this is useful. The trouble was, if you enabled APM while the laptop was in the docking station, the laptop would freeze. So there's a very long-standing configuration option that says: do you want to enable APM? Because you don't necessarily need to do the enable because it could already be enabled. But this particular laptop, even if it was enabled, if you did the enable, it would freeze. So there's a compile-time config option for it. *20m, 26s*

At one point, we had ten compile-time options for the APM code in the kernel. And I think all but one of them were for BIOS workarounds. So if you had this particular laptop, you enabled these options; if you had that particular laptop, you enabled those options; just so your machine wouldn't die, which is not good. *20m, 48s*

Next thing we did was software power down. In 2.0 and before, what that did was we had an option that said: power off at shutdown. You did it at compile time. Anytime somebody did a shutdown to a halt, it would power off or not, depending on whether you configured the option. In 2.1, Linus decided, or someone decided, that it was a good idea to split halt and power off into two separate things; and they were right of course, but I got all of the bug reports saying: I do a halt and it doesn't power off any more. And I said: well what did you do? And he said: I only upgraded my kernel. Good. The speaker laughs. *21m, 34s*

So we then got rid of the power down on halt configuration option, so one goes away, that's good. But you had to pass -p to halt, otherwise it didn't power down. And of course, the most popular distribution at the time, Red Hat, didn't pass -p to halt by default. So as soon as everybody upgraded their kernels, I got all these bug reports. I think I answered the question probably several hundred times in a year and a half, by saying: just put -p. Oh, they say, it works. Good. *22m, 06s*

The next thing that came up was, about a year ago, a year and half ago, one of the BIOS writers released a new BIOS. And obviously they tested it under Windows, which is fine, but they didn't test it on anything that stayed in protected mode when it halted or tried to power off. And so all of these people were getting: they would go halt and it would panic, after it, right at the end. It's not a big problem because everything is unmounted at the end, but they would get this panic. I thought, oh good. So we invented the real mode power down option, which meant switching the kernel back into real mode before we powered down and that solved that problem. We actually got some of the BIOS manufacturers to fix their bug and we got at least one of them to acknowledge it on their web page, but the rest of them called it a Linux problem. The speaker laughs. *23m, 01s*

The next thing we wanted to do was to improve the idle loop, so when you're idle you can actually save power because: what do you want to do when you're idle? Nothing. And when you're doing nothing, how much power do you want to absorb? As little as possible. This actually started before APM came along. Linus came up with this brilliant idea, he came up with halting. When you're not doing anything, don't do anything. Windows still hasn't figured this out; when you're not doing anything, you do something. The speaker laughs. I don't know what they do, but they do something. *23m, 38s*

On Intel, anyway, there's an instruction that says halt and it does exactly that, it just stops. And it doesn't do anything until it gets an interrupt, which is useful. If there's nothing to do, there's no point in doing anything. This becomes even better because on the 486SL chips, the later ones, the power managed ones, and all of the Pentiums, when you halt, it actually stops the clock because the insides of the chips are static, so you can do this. And when you stop the clock, the power usage of the chip drops through the floor pretty well. So doing a halt actually turned out to be a useful thing and people suddenly discovered that all their laptops were running quite a lot cooler when Linus put the halt in. And their fans didn't turn on so much in their desktops; all sorts of things were really good. *23m, 23s*

The next thing you want to do is, you can actually tell the BIOS that you're idle, the APM. And when you tell the APM BIOS you're idle, it will do things like say: we're not using this and power down all these things. Or it will say: let's run the CPU slower. Which is actually not useful if you're halted, because the CPU is not running at all. But it can be useful if you're doing other things, like Windows does; so you run the CPU slower. The problem with that, of course, is that you have to tell the thing when you're not idle anymore, otherwise your CPU runs at like... some of these BIOSes actually lowered the CPU to one percent of its rated speed. So you can imagine, you've got your 100MHz Pentium, suddenly it's running at 1MHz. Which is why my kernel compiles were taking so long. *25m, 10s*

In 2.2 and before, we basically had a hook into the idle loop, so that if we had APM enabled, we would just tell the BIOS that we're idle. In 2.3, Linus thought it would be a good idea if we had a separate power management idle loop, so (he) we invented the kernel APM daemon and I started getting bug reports about this process that was using all our time, called kapmd. And if you sat there just running top on a 2.3 kernel, the top process, if you're not doing anything else, will be kapmd and it will be using like 85 or 90 or 95 of your CPU time. These people were worried because it was idle: why is it using all of the time? Well actually, it's just that the time is getting accounted to that process. It's not doing anything, it's the idle loop. *26m, 12s*

(What else have we got?) One of the biggest absorbers of power on these things is not the CPU. The CPU is one of the biggest ones, but not the biggest. Up to 40 of the power (in your CPU) can be used by the backlight on your LCD and the LCD panel. So a good thing to do is when you're not looking at the monitor, turn the backlight off. So we put a hook into the virtual console blanking routine that would actually turn the backlight off. Because just blanking the LCD and leaving the backlight on doesn't save you anything, almost, you're still driving all the pixels and the backlight is actually a little neon light; they used to be little neon lights, I'm not sure what they are now. *27m, 01s*

So we do that. We have /proc/apm, which basically tells you status, so you can look in there and it will tell you that you've got this many seconds of battery life left, if the BIOS knows and tells you. And we have /dev/apmbios, which is an interface between user mode and the kernel device driver, so you can do things like tell the kernel to suspend now. Or you can get told when the BIOS wants to suspend. *27m, 30s*

This is a dead end. The last version of the APM spec came out in 1996; that's four years ago. All of the concentration and energy seems to be on ACPI now. I'm still doing work on APM, occasionally; I still get bug reports. Lots of people's laptops are still only running APM; lots of desktops don't implement ACPI yet. Well, any new desktop or any new laptop you buy will support ACPI, however the ACPI in Linux is not really up to it yet, so APM is not quite dead, but dying. *28m, 11s*

ACPI... ACPI requires a user mode daemon. Otherwise, it doesn't do anything. Good start; that's okay. That's exported... all of the policy is now out of the kernel. The kernel is just an interface to the ACPI and the policy is now in the user mode; the daemon decides when to go to sleep, what devices to power off, it can do all that sort of stuff. *28m, 36s*

It currently is capable of doing software power off. It currently will suspend on some laptops because I've seen one that it did it on, but not mine. So it's still getting there. It actually uses these different CPU power states in the idle loop. So the longer you stay idle, the lower the power you use. So in this sense, it probably does a better job of idling than APM does. ACPI is definitely a work in progress. There's lots of work going on. As I said, the ACPI guys recently got a big patch into the kernel. I tried it out yesterday and it didn't do anything different than the old stuff, which was this big and now it's this big. *29m, 26s*

This is, in theory, the way of the future. However, it's pretty large. *29m, 39s*

The speaker calls for a question from an audience member.

Explain what sort of things ACPI breaks on right now?

Breaks on? Ahh, sorry. This is read: the way to go; as opposed to it's got a way to go. It does also have a way to go. For instance, there's no implementation of any policy. It's very simple at the moment, it knows how to power off the machine. On some laptops, it will actually suspend. There's a proc entry... /proc/sys/acpi/sleep and if you write anything in there, the machine should go to sleep. This laptop doesn't do that, but I've seen one that did. *30m, 21s*

There is an unintelligible question from the same audience member.

It uses the different CPU power states in the idle loop, so it does that as well. The infrastructure is now in there to do power management of all of the devices, except there's no user level interface to do it at the moment. I'm not actually on the ACPI mailing list, so I don't actually know where they're going; all I can tell you is what I can see from their released code. *30m, 50s*

There's a tiny bit of application control. The acpi daemon exports a socket which you can connect to and send commands on. The only command which I think is implemented is: get me the battery status. Which may be useful; at least you know your battery is dying. *31m, 13s*

Driver APIs... I've added this just recently; this being a technical conference, I thought I should talk about something technical. What we need from driver writers at the moment is for the drivers to cooperate in the power management of the system. Because if the drivers don't cooperate then we don't save any power; in fact, in theory, if a driver refuses to suspend, you shouldn't actually suspend the system, because when you resume, the driver won't know and may well need to reinitialize its hardware and won't know that. The classical example is when you're in X and you suspend, and you come back, you're not in X anymore. *32m, 05s*

So what we need is... well that's more of a user mode issue, but the same is true of like sound cards; a lot of sound cards don't recover from being suspended; Ethernet cards, it goes on and on. There are currently, under Intel, three APIs for drivers to interact with the power management. You can talk directly to the power management; under 2.2, that was the only way, no that's not true. Under 2.2, as a PC card, you could also interact with the power management. Under 2.3, you can, as any PCI device can just register itself with suspend and resume callbacks, so it will know what's going on. And all PC cards can register for device events as well, which is actually very useful when you think about it. *33m, 01s*

Now the direct way... Under 2.2, we had APM register callback, because APM was the only way on Intel. Basically, you said to the APM code: here's a function, just call it when you get an event. So the APM idle loop is sitting there going: are there any events, are there any events; and the BIOS says: yeah, we want to suspend now. Any driver that had registered with the APM will get this callback saying: we want to suspend. And the driver can say: just let me do a few things, okay, go. Or it could say:

no, you can't do that now, we're waiting for something and we're halfway through paging in a critical page, don't do it. There is also a user mode equivalent of this; if you listen to /dev/apmbios, you would get an event out of there, and you have got the choice, you could say: that's okay now. If you tried to say: that's not okay now, it wouldn't do what you wanted to do anyway. This is something I want to fix. *34m, 05s*

In 2.3, the ACPI guys invented pmregister, which was useful. It means we're now independent from the underlying power management. It is effectively the same, though; you say pmregister and you get a callback routine and it will call you back for certain events. Basically only suspend and resume at the moment. But it does allow you to do anything you need to do. *34m, 33s*

pmaccess: if you're going to access your hardware, you should call pmaccess to make sure your hardware is awake before you access it, otherwise bad things happen. And pmdevidle is a hint to the policy to say that this device is now idle, you can power it down now if you wanted to, but just so you know it is idle. Okay, so these being what you want to register; what you want to find out about things. *35m, 05s*

The PCI device registration... When you register a PCI device driver, two of the things you can do is give it a suspend/resume hook so whenever the power management policy driver, whatever that is, decides it wants to suspend the machine, all the PCI devices get told about it, if they've registered these hooks. This is actually very useful because one of the things you can do to a PCI bus is turn it off. If you turn the PCI bus off while the devices are still alive, the devices aren't really happy. So what you've got to do is do things in the right order. *35m, 45s*

The other thing is that PCI busses can be chained, so this PCI bus can hang off of that PCI bus and if you turn this one off first, you're in trouble. So the PCI subsystem takes care of this and goes through, effectively recurses across all PCI busses, bottom up, tells all of the devices, tells all of the busses, then turns off the top-level PCI bus, if there is such a thing. So you can actually, in theory, go around and turn off the particular PCI devices, or you can turn off a whole bus if you're not using it. All this just saves power in good ways. PCI devices should also register directly with the power management so they can do pmaccess and pmdevidle calls, because they're good hints to the policy, so we know what's going on. *36m, 37s*

The PC Card driver registration is very similar; if you're a PCI device and a PC Card device, i.e. you are a Cardbus device, you don't need to do this because you're registered with the PCI subsystem anyway. But if you're not, if you're a regular sort of PC Card, then this is useful because the PC Card subsystem has sort of a global callback thing, which allows you to tell drivers things like: your card has gone away; or we want to go to sleep now. And again, you want to know that because you want the device driver for the card to know that you've gone to sleep before you turn the power off to the PC Card controller, the PCMCIA controller, otherwise the driver may just die badly the next time it tries to access the card. And it does things in the right order, so it turns the cards off before it turns the PCMCIA controller off as well. *37m, 41s*

As well as this, which I haven't got a slide for, there's a user mode interface to APM, at least. The ACPI one will have an equivalent. And I'll talk about PowerPC in a minute. Which allows any user mode process to connect to /dev/apmbios and be told about all of the events, so you know when the BIOS wants to suspend; you know when the user has pushed the suspend button. There are other things that you know; there's a standby mode which is sort of half-asleep, you can find out about that. This wasn't being used by a great many things, except a thing called apmd, maintained by Avery Pennarun over here. This is what you do, you write the infrastructure and you leave policy to somebody else. And I've got somebody else, so this is all right. *38m, 31s*

Avery comments on this matter.

I leave it to the distributions to decide exactly what they want to do with the power.

Yeah, well, you've pushed it up another level. Somebody else will do the policy, that's okay. But apmd allows you to do arbitrary things, because it basically calls a shell script on each event and the shell script can do arbitrary things, like it can do a software eject on your PC Cards, because there are some PC Cards that don't like being suspended, they need to be ejected when you do a power off. You could also do things like switch away from your X server before you suspend and then switch back afterwards, because suspend works in text mode, but it doesn't work in graphics mode. *39m, 11s*

This has become less necessary now, because I got this mail from one of the XFree 4 developers who said: we've discovered this APM stuff; it would be really useful if you did this little patch. So I did the patch and so now XFree 4 knows about APM, so now when you suspend, the X server gets told that you're going to suspend, so it can save all this data, the graphics, and do what it needs to do. And when you resume, more importantly, it can put it all back again. So suddenly everybody's X servers work. *39m, 46s*

There is a question from an audience member.

There's a difference between APM and ACPI support for devices, right?

Absolutely.

The code they run to turn themselves off, is it written in machine language?

It can be, or if you know enough about the device, a lot of devices these days have power management things you can do to them. Okay, so in particular, the PCI spec defines a generic way of turning PCI devices on and off, or changing their power level. But in the case of a device you don't know about, yes, you basically interpret the AML associated with the particular power off method for that device. And that's why you need the interpreter in the kernel, because you may have to do this when you can't load modules or when you can't go out to user mode, like it's your disk controller that you're booted off and you need to do it. Yeah, so it is a bit different and it's actually reasonably hard and a pain. *40m, 59s*

We've actually found a good use for the AML code, though, because you can interpret it by hand, which means: if the people that put together your laptop bothered to build the ACPI stuff for your laptop, you can find out about interesting devices on the laptop that they won't tell you about without an NDA or they won't even tell you about at all, but they still allow you to enable and disable devices and power manage them, even without knowing what it is, which is useful. *41m, 27s*

But the XFree thing was actually useful. Unfortunately, it's already out of date because APM is going away and ACPI is taking over. But ACPI doesn't currently define a user mode interface, so we're sort of stuck at the moment, which is why APM still runs on a lot of laptops, I guess. *41m, 47s*

Other architectures, quickly... besides these, there's also SPARC, which I'm told you can power off in software, but I have no idea how to do that and I've got no idea if there's anything else useful. We know that happens because when you install Solaris on an UltraSPARC, one of the questions is: do you want this machine to turn off after thirty minutes of idle time? You do this at installation time, months, and you say yes, and you leave the machine idle for thirty minutes and it turns itself off. *42m, 20s*

There is a comment from an audience member.

There is an /etc/powerd.conf, but I'm not sure what it does.

I'm not a Solaris person. We just discovered this because we came in one day and the machine was powered off. The audience laughs. *42m, 32s*

On the PowerPC, laptops at least, Paul Mackerras, who's the maintainer, did his own thing. In a sense he's lucky, because there's basically one set of hardware. There's a PMU, a Power Management Unit, and I assume there's specifications for it. *43m, 00s*

There is a comment from an audience member.

Apple may have some specs internally, but they haven't released it.

Paul figured out how to deal with this device. And so, the PPC port does some power management. Basically, it has a user mode daemon that does all the work, like ACPI does. Any applications that want to interact with the power management interact with that user mode daemon. And there's a restricted set of hardware, which makes managing that easy. However, I suspect Paul is going to discover that hardware gets a lot harder soon, because there's all these Power PC machines out there with PCI busses and people are putting cards into them. And it's going to get a lot harder. *43m, 47s*

One of the things we need to do is to integrate the power management from the different architectures, so that, as far as the drivers are concerned, it's the same. At the moment, it is different on the PowerPC. *44m, 01s*

Transmeta Crusoe, this slide was written a while ago and we didn't know much about it. We still don't know a hell of a lot about it; Dave Taylor of Transmeta can tell us a lot more. The speaker laughs. One thing we do know, though, from the whitepapers, is that it can do not only speed changing of the CPU, but voltage changing of the CPU, which tends to save you a lot of power. The hardware, again, is fairly restricted, so they understand it well and can do what they like with it. My impression was, and I'm not sure this is correct, that the first pass, they did an APM BIOS. *44m, 39s*

There is a mostly unintelligible comment from Dave Taylor of Transmeta. The gist of his comments appear to be that they don't support CPU throttling yet, also known as duty cycling, although they do change the frequency. Control of this is done by MSRs and he believes they have already put together an application for control of this, though it has not been integrated into the kernel yet, though they've really just started checking in the crusoe tree. ED: The preceding section of this transcription represents a best-guess. Readers are encouraged to seek validated information regarding power management capabilities from Transmeta, rather than relying on the statements here. *45m, 21s*

We'll watch for it. I did notice that you now recognize Crusoes, that went in there. So they now recognize a Crusoe and say: we've got a Crusoe, good. The speaker laughs. Again, it's all software controlled. There's a couple of whitepapers all about it on the Transmeta site. I think, Dave you said you were going to switch to a more ACPI style. *45m, 44s*

There is a mostly unintelligible comment from Dave regarding Crusoe and ACPI. Yeah, we've been talking about ACPI because we thought APM was dead. And we've been pushing for this very hard because we have all these web slates. You mentioned earlier that the backlight is sometimes up to forty percent of the power; but for us, it is up to two- or three-hundred percent. So it's a terrifying problem... The remainder is unintelligible. *46m, 24s*

One thing you discover, though is that every time your laptop comes out, for instance, it has a new video chipset that you can't get the specs for. So I assume you're going to have the same problem with the web pads and things; it will just be a new video chipset in every time. I don't know; they like to put the gee-whiz latest thing into these things and it's just bad. *46m, 44s*

That's about all I want to say. I've got a list of references here; these slides will be available on my website, which is the top one. Not at the moment, but they will be. The APM daemon website, that's Avery's website. *47m, 03s*

It's spelled wrong, there's only one p in apenwarr.

There's a correct link off my website; it works, I've tried it. The ACPI website; this may not be actually correct anymore, but it will get you to the right place. If you want the documentation, Intel doesn't actually publish the documentation on their website anymore, which was really exciting for me because that's where I used to link it from. I had about three dozen people tell me that the link broke, so I went searching and it's just not there, as far as I can see, so you've got to get it off the Microsoft site, if you want it. The AMP there is not a typo, I don’t know, that’s what they did. *48m, 00*

What about other embedded chips, that have registers and things to let you slow the processor down; is there the ability to add support for that?

Of course we have the ability; we have this generic power management interface now. The hard bit is getting the devices to do things at the right time, and we’ve got that as a generic interface. So devices can register and say: I want to know when things are happening to the power levels. But to do the actual CPU throttling or changing the speed of the chip and things, that’s all up to the particular architecture, hardware. I just don’t know about any of the other architectures and I don’t know if any of them do any throttling at all. I mean it’s mainly come from the laptops, webpad sort of arena. And almost all of the laptops are Intel. *48m, 53s*

Is there some mechanism to handle the change in clock rate, so the jiffies...

No; this is an interesting thing about it. There are some laptops which, if you boot them and their batteries are low, they actually boot into low-speed mode. Like one of the IBM ThinkPads will boot into 100MHz instead of 400MHz. And if you do that, it does the BogoMIPS calculation at 100MHz and gets this number, then you plug the power in and it speeds up. So all the udelay loops are a quarter of what they need to be, all of the delay loops are a quarter of what they need to be, and this causes really interesting behavior. There is no support in Linux for changing the clock rate and coping. It’s okay if you do the BogoMIPS calculation at full speed, and you only slow down, it’s probably okay because the delay loops only get longer. *49m, 43s*

What about time and date when you slow down your clock?

That all works, that’s okay. That’s one of the things we do, is set the clock back. Well, I don’t know. It’s supposed to work. *49m, 57s*

I’ve got this old 486 laptop that you can change the speed between 66MHz and 33MHz with a keystroke.

Oh good. Start at 66MHz. It’s been discussed. The only thing you can really do is recalculate the delay loop. *50m, 20s*

Immediately.

And you don’t know that the clock rate has changed.

On these embedded systems, you would...

If you know, that would be useful, except on Intel systems, anyway, you've got to turn all the interrupts off and basically run for a second or so with all of the interrupts disabled and figure out how many instructions you can execute in a second. Which is a pain, and you don't want to do that at arbitrary times. *50m, 49s*

The speaker calls on an audience member for a question.

What's the granularity of the udelay you were talking about?

udelay? Microseconds.

So could you not use unintelligible?

The kernel's been changed during 2.3 to use the clock counter. The problem with that, of course, is that Intel's TRC changes speed with the... I think Transmeta guaranteed that it wouldn't change, is that right. *51m, 17s*

Well we faked it so it wouldn't change.

Intel's changed. One of them, Cyrix I think, has a bug so that its possible to read the TRC, which is the cycle counter; you read it twice and it's possible for the difference to be zero. Which is not a good thing. You wouldn't imagine that's possible given you really can't execute the two reads in less than a cycle, but they did it. Basically if you went to sleep between the reads and came back, they didn't update the cycle counter. The speaker adds, with sarcasm. This is good. *51m, 47s*

The speaker calls on another audience member for a question.

How would you go about debugging a bug involving PCMCIA suspend and the BIOS. I mean I've got this Sony Vaio laptop which doesn't work after a suspend...

Oh, you've got a Vaio that doesn't work. Oh, there are Vaios that do; most of the Vaios work.

There are many that don't. unintelligible interrupt controller after suspend. Something screws up the interrupt controller. How would you actually start to debug this?

You've really got to guess, you've got to get specifications for the devices and see what they're doing. Try things and see. For instance, there was one machine where when you suspended and resumed, the tick clock was set back to 18 Hz like Windows has. And so we now have a patch in which reinitializes the clock back to 100 Hz again. There are just... you've just got to find these things out and try to guess what's going wrong. *52m, 54s*

When did that patch go in?

A long time ago.

Because my laptop stopped working a long time ago, for sort of exactly the opposite...

Well, it's the same code that really initializes the thing in the first place, so I'm really not sure that would be a problem; that it would be the problem, there's lots of possibilities. *53m, 19s*

The speaker calls on Dave Taylor of Transmeta for a comment.

I just wanted to answer the question about debugging power management. One of the things we, first of all sorting through it is really hard. And the longer answer is one of the things you've got to do is to take whatever system you have and call the manufacturer and say: can I have this PCI card format? And then you can debug it on the desktop. Once you can debug it on the desktop, you can plug in a PCI bus sniffer and the bus sniffer can tell you all kinds of neat things about whether the Southbridge is inserting unintelligible, so you can find out whether your PCI space unintelligible. *54m, 08s*

Basically, you need a lot of hardware around you to figure it out. Because the software is really out of control here; there's nothing we can do. Especially in APM mode because we don't have access to system management mode; if we had access to system management mode, we could in theory build a debugger inside that could watch what was going on, at least partly, it wouldn't be able to do what a PCI sniffer can do, but at least it would give you somewhere to go. But you'll find with any machine running APM, there's no way you can get into system management mode, because as soon as you jump into system management mode, you're in the APM BIOS and it's doing things. *54m, 43s*

These are just the last of the references. This Intel's SL architecture book; I wouldn't recommend anybody read it, it's just where I first found out about the system management mode and stuff. And documentation/apm.txt, all of the driver writers should read this. The first thing it says is: if you're writing a device driver, it must do power management. The speaker laughs. Full stop. Otherwise, we just can't do things. At the moment, it's a bit strange because if you're a PCI device and you don't have a suspend hook, we will still turn the power off on the PCI bus on you, which means when we come back, you may or may not recover. So everybody should do it. *55m, 26s*

The speaker calls on an audience member for a question.

Would it be possible to have sort of have power management operate at a higher level, like always call device close and device open on, say, a network interface? Instead of having every single network interface have a special suspend and resume.

There is a comment from another audience member.

Some of the network drivers already do that.

The original audience member expands his question.

That's what I'm saying, though, why should I have to change every single network driver? Why not add it on at a higher level and call every network driver and say close and open on suspend and resume.

Because close and open may not be sufficient. There are some devices which cope with suspend and resume okay and don't need reinitialization. There are some devices which basically have to be reinitialized as though they were just powered up, or they just don't come back. *56m, 11s*

The original audience member continues.

I used the network card as an example because I know drivers are supposed to reinitialize when you reinitialize and when you do an ifconfig up.

Not all of them do that. *56m, 23s*

It's a bug more than anything. So you should be able to catch most problems just by downing the interface, right?

You should, most, yes. Most is the problem, right? *56m, 33s*

It's better than nothing, though.

It's the same with PCI devices; there's a generic way to control the power management level of a PCI device. The PCI spec defines a generic way, but the code in the kernel, if you look, says they don't define a suspend and resume hook, we're not going to do the generic way because some PCI devices just don't cope. Even if they say, there's a capability bit in the PCI bus that does power management, and the things we'll do... even if they do say that, some of them don't cope and so basically we've got to work around all of this buggy hardware. The best place to do it is in each device driver because in theory, each device driver knows about the bugs in the hardware, hopefully. *57m, 16s*

There is a comment from an audience member.

I've heard above the bugs in the hardware, sometimes the BIOS is the one that initializes the hardware, so if you go into a power down state and then come back to a power up state, the device driver may not have enough knowledge to bring it up to a working state.

There's just too many unknowns. *57m, 38s*

The speaker calls on another audience member for a question.

User mode ACPI, what's missing?

User mode ACPI.

We already have acpid and acpictl.

Yeah. The last version I've got, which was from May, acpid merely initialized some things so the kernel could do the sleep modes and would listen for interactions from other user mode things saying: what is the battery status. And as far as I know, that's all it did. *58m, 19s*

It got upgraded in early July when they put the code in the kernel.

That's what I need to look at, because as I said I haven't had time over the last couple of months to actually follow-up with what's going on with them, so I don't know what's missing. The ACPI control program that came out in May wouldn't even compile, so I didn't get very far with it. In theory, the infrastructure is now all there. So they can just sort of power forward and get things done. And I would be more than happy for them to get it all done, because then when people come to me and say: my APM doesn't work; well I'll tell them to go and try ACPI. But for the moment, I can't tell them to go and try ACPI, because their machines won't even suspend. *59m, 04s*

There is a mostly unintelligible comment about having to fix all of the BIOSes with bad ACPI.

Well, there are workarounds in the ACPI code for BIOSes they know are broken. And so, we can try. *59m, 13s*

The speaker calls on another audience member for a question.

In the 2.0 kernels, I found a program that was called, I forget what they called it, a power management version of the bdflush program, that would basically spin down your hard driver and then refuse to flush any of the kernel buffers until the hard drive was spun back up again for a read. I haven't been able to find anything like that for 2.2; have you heard of anything like that?

No. One thing I've done recently, though, is talk to the IDE maintainer and say: why don't you hook the APM stuff? And he wrote back to me recently and said: did you fix all your problems with the IDE and I haven't written back yet, but my reply will be: the problems with the IDE aren't my problems with the IDE. The problems are the IDE problems with the power management, because for instance the IDE stuff doesn't know when you suspend and resume at all, because it doesn't hook the stuff. *60m, 08s*

And in particular, if you compile APM into the kernel, the only difference to the IDE driver is it will wait 30 seconds for the command to complete because the disk may have to spin up, whereas it normally waits like 5 milliseconds or something. But it has no knowledge of the power management. If it had knowledge of the power management, then it could do things like blocking the device queues until you really wanted to do something. And the disks could then spin down and they wouldn't just spin up because somebody's just written a log message, like the APM daemon has done a log message and it has to spin the disk up again. Or the APM device driver has done a log message because it's in debug mode or things like that. It would be nice for the disk subsystems to know about what's going on. Because you could just block device queues, and that's a useful thing. *61m, 00s*

There is a comment from an audience member.

When the pulled bdflush into the kernel for 2.2, we sort of lost the ability to do that; easily, anyway.

Well, you can patch the kernel; we've got the source. The speaker laughs. Anything else? Done, good.

The audience applauds. *61m, 24s*

3. Additional resources

3.1. Presentation materials

The slides used in this presentation may be found at http://www.linuxcare.com.au/sfr/pmtalk/index.html

3.2. APM for Linux home page

Details regarding the current APM implementation in the Linux kernel, as well as information regarding APM itself may be found at http://www.linuxcare.com.au/apm

3.3. apmd

Details regarding the apmd application and project may be found at Avery Pennarun's site, located at http://www.worldvisions.ca/~apenwarr/apmd

3.4. ACPI for Linux home page

Details regarding the ACPI for Linux project may be found at http://phobos.fs.tum.de/acpi