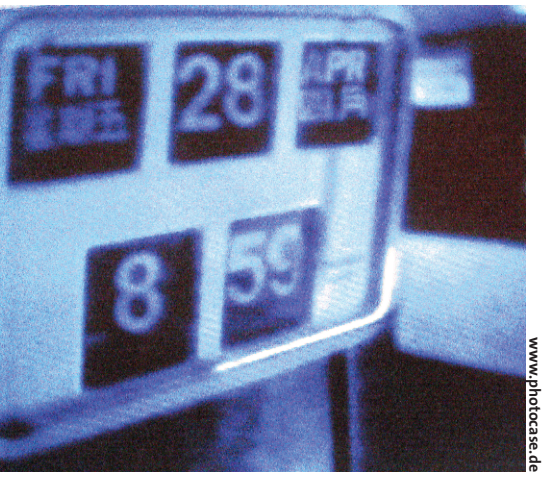

www.photocase.de

Programming a wakeup service for your computer

Wakeup Call

Switching computers that do not need to run 24 by 7 on and off as required can cut your power consumption. In this article, we will be looking at three different methods of programming a wakeup time under Linux without needing to set the time manually in the computer's BIOS. BY MIRKO DÖLLE

Servers need to run 24 by 7, as do RAID systems. Most users tend to leave their desktops running when they leave the office in the evening, although computers are rarely needed for more than 10 hours per day. For special applications, like fax servers that send out fax messages at 2 o'clock in the morning, why not boot the machine just before two and switch it off again for the day until the next nocturnal session is due?

Implementing these scenarios is quite simple. All you need is a time switch for a couple of dollars to switch the machines on and off as needed. Things start getting expensive if, as a normal user, you do not have access to the time switch in the data center. In this case expensive power switches and a computer-based controller are typically the only viable solution.

The BIOS on your computer's motherboard will typically allow you to wake up your computer without additional hardware. More or less any modern machine should have a function for time-controlled booting, and let's not forget the additional "Wake on LAN" and "Wake on Ring" functions that allow a desktop to wake the server whenever it needs to do so.

The problem is how to change the wakeup time for time-controlled power up on Linux – different manufacturers and BIOS revisions mean applying different approaches as no one method is ideal for all. For example, ACPI wakeup does not work on many motherboards due to different implementations of the standard. In contrast to this, NVRAM wakeup can use non-volatile memory, that is the memory where the BIOS stores its data, to allow direct editing of the wakeup time. Of course, this assumes that you find the right memory location, and that your BIOS actually notices that the values have changed. The third method, *settime*, uses a simple trick. The BIOS always wakes up at the same time and date, and the computer clock is reset to a specific date just before you shut down the machine.

ACPI – the Secret Weapon?

ACPI is probably the easiest way to switch on a computer at a preset time, that is, assuming that the kernel supports your choice of motherboard. If this is the case, then you can simply store the wakeup time for the system in */proc/acpi/alarm*:

```
echo 2004-08-02 20:15:00 >/proc
/acpi/alarm
```

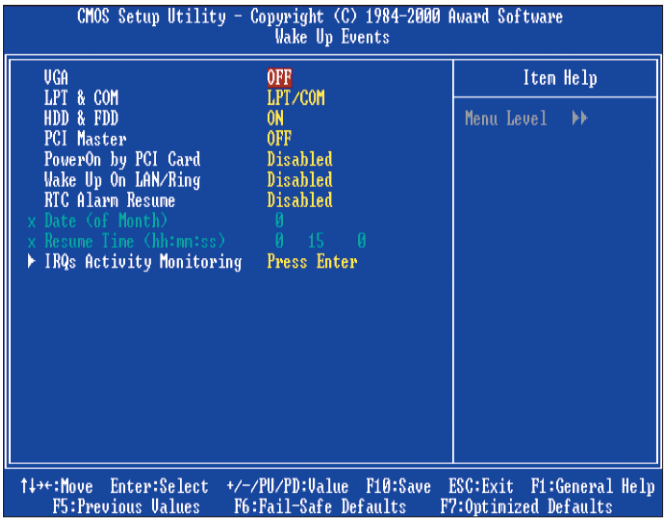

Figure 1: Most computer BIOS chips have a wakeup function called *RTC Alarm Resume* or something similar.

The kernel transfers the time directly to your computer's RTC (real time clock), but not the date. This leads to the computer waking up at the same time every day, and not just on the given date.

The problem is that there is a more-or-less standardized method of transferring the time to the RTC, but no such standard for the wakeup date. Consequently, the *acpi_system_write_alarm* functions in *drivers/acpi/system.c* for the kernel 2.4, or *drivers/acpi/sleep/proc.c* for the kernel 2.6, only support the alarm time fields.

Out of Date

Kernel developers Andy Grover and Paul Diefenbaugh have already prepared the way for the full functionality, but they need a usable fixed ACPI table (FACP). If you check out the comments in the kernel code, you will note that the current FACP tables for motherboards are unserviceable, and this has led to the code block being permanently disabled.

The various motherboards pose a variety of problems. Some motherboards do not pass the time correctly to the computer's RTC, storing the last boot time instead of the desired wakeup time. Others immediately crash when an

attempt to write to */proc/acpi/alarm* is made. ACPI wakeup works with many of the current crop of motherboards. You can checkout the distribution section on the LinVDR (mini video recorder) distribution homepage for a non-exhaustive list [1].

Topsy-Turvy

If you want to use ACPI to wake up your computer, you need to ensure that its BIOS settings are correct. For years now most computers have had BIOS functions called *Wake on Timer*, *Resume on Alarm*, *RTC Alarm Resume* or similar variations on the same theme. This said, users were expected to set the time in the BIOS. ACPI wakeup uses a similar technique, but it does not use the BIOS function. This is why you will need to **disable** the BIOS wakeup function on most motherboards if you want to use ACPI. Also, note that the time for the next time-controlled power up will not be displayed in the BIOS.

Yet another hurdle: there are a few motherboards, such as the Asus A7V133, that do not take kindly to a shutdown script using *hwclock -w* to synchronize the RTC with the system time after entering the wakeup time. The computer simply does not wake up again. You need to set the *--directisa* flag when calling *hwclock*, or as an alternative, you can set the wakeup time after calling *hwclock*.

Groundhog Day...

As previously mentioned, ACPI wakeup tells the computer to wake up every day at the preset time, and this does tend to restrict the usefulness of the function. ACPI wakeup would be quite useful for a fax server that automatically processes fax messages every night before switching off again. ACPI wakeup is also quite useful for PC video recorders (PVRs), such as VDR for example. Most channels update their electronic program guides (EPGs) between midnight and two in the morning, so booting the machine at two o'clock would allow you to perform an EPG scan to support automatic timer programming.

However, a print server that boots at 9 in the morning and powers down at 7 in the evening would be up and running for 10 hours on weekends. An additional routine in the startup scripts that recognizes weekends and public holidays could shut the machine down, but this is only a stopgap solution. Although you might need to use the print server on the weekend, it would still shut down, even if you booted it manually.

A script that checks for weekends and public holidays when shutting down the computer, and programs the wakeup time and date, is a far better solution. NVRAM wakeup and the *settime* method provide the answers.

NVRAM Wakeup

NVRAM wakeup uses the BIOS settings stored in non-volatile RAM (NVRAM). The *nvram* kernel module allows Linux to access a maximum of 128 bytes of non-volatile memory.

To get this to work, you need to compile the *nvram* kernel module, and also create the character devices */dev/nvram* with a major of 10 and a minor of 144, */dev/rtc* with a major of 10 and a minor of 135, and */dev/mem* with a major of 1 and a minor of 1. You also need to modify the */etc/modules.conf* module configuration file to tell your computer to load the NVRAM module when */dev/nvram* is accessed or immediately on booting.

Listing 1: nvram-wakeup.conf

```
01 ##################################
02 ##  motherboard autodetection information:
03 ##
04 ##    - motherboard vendor:    ""
05 ##    - motherboard type:      "VT8367-8235"
06 ##    - motherboard revision: ""
07 ##    - BIOS vendor:          "Award Software
   International, Inc."
08 ##    - BIOS version:         "6.00 PG"
09 ##    - BIOS release:         "04/07/2003"
10
11 addr_stat         = 0xD2
12 shift_stat        = 5
13 addr_day          = 0xD8
14 addr_hour         = 0xD9
15 addr_min          = 0xDA
16 addr_sec          = 0xDB
17
18 upper_method = VT8235_37
```

Using BIOS Functions

There are no tricks to watch out for when building the program from the source code (available at [2]). The default installation path is */usr/local*, but admins can edit the *Makefile* to change the path to */usr* if needed. If the */dev/nvram*, */dev/rtc*, and */dev/mem* devices do not exist at this point, *make devices* will automatically create them.

In version 0.96, the *tr* statements in lines 62 and 75 of the helper script *guess-helper.sh* do not work:

```
answers=`$echo $answers | ➥
tr [:lower:] [:upper:]`
```

Fortunately, the script worked fine after adding four backticks

```
answers=`$echo $answers | ➥
tr '[:lower:]' '[:upper:]'`
```

For most manufacturers, the way they allocate the NVRAM is a big secret – you might have some difficulty finding out where the date and time are stored for lack of documentation. What's worse, the memory locations tend to be a moving target that changes whenever you update the BIOS.

The NVRAM wakeup sources include specs for a number of boards. These details come courtesy of the *guess-helper.sh* script. Running *nvram-wakeup* with the *-D* flag launches the program in debug mode, and tells you if your motherboard is known. If *nvram-wakeup* tells you that

```
nvram-wakeup: Your motherboard ➥
is currently not supported.
```

however, you will need to run *guess-helper.sh*.

The *guess-helper.sh* script tries to locate the position of the NVRAM memory fields. You will need to boot your computer at least four times to do this. The first time you boot, you need to disable the BIOS wakeup function, set the time to the 31st of the month at one

second to midnight, and then call *guess-helper.sh*. Then set the wakeup time to 11th of the month and the time to 12:13:14, then to the 1st of the month, midnight, and finally disable the wakeup function again.

Each time the wakeup point is called, *guess-helper.sh* compares the content of the NVRAM with the other calls to establish the memory locations for wakeup points. *guess-helper.sh* uses two methods to do so: access via */dev/nvram* and direct I/O address access. This results in two configuration files, *nvram-wakeup.conf* in the *guess-nvram-module* directory, and *guess-directisa* below *root*'s home directory.

Positioning Information

A quick look at the configuration files shows us where comments about the motherboard, and address information are stored. In the case of the Elito Epox 8K5A2+ board, the */root/guess-nvram-module* file was empty apart from a few comments, whereas the */root/guess-directisa* contained the address entries shown in Listing 1.

If both configuration files are empty, you can assume that *guess-helper.sh* was unable to locate the addresses.

You need to tell the program the path to the configuration file, using the *-C* parameter to do so, whenever you call *nvram-wakeup*. Also, the *-A* parameter is required in this case, as we will be using the I/O addresses for direct access rather than the */dev/nvram* device. Both these parameters can be left out if you have a motherboard that *nvram-wakeup* supports.

Trial Run

For our trial run with *nvram-wakeup*, we need an arbitrary wakeup time in the BIOS; we will be using *nvram-wakeup* to validate the results.

In the following example, the BIOS is set up to wake up the computer at 20:15 on the 15th of the month. The date for the *nvram-wakeup* call also needs to consider the time vector between local time and UTC, and allow about five minutes for the computer to boot. Figure 1 shows the output for *nvram-wakeup*. The top section of Figure 1 shows the wakeup data that *nvram-wakeup* located in the NVRAM; the bottom value shows the data that would have been program-med without the write protect parameter, *-N*. The values match, so we can assume that *guess-helper.sh* really has located the right memory cells.

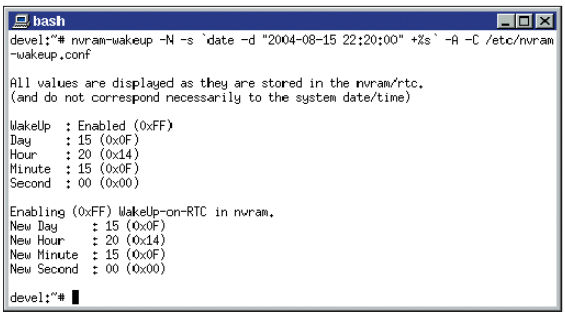

Figure 2: For a successful trial run, the values displayed by *nvram-wakeup* need to match. The wakeup time set in the BIOS is shown at the top; the bottom value shows the value as parsed by the program.

The next step is to set a date about 15 minutes in the future, and then shut the computer down:

```
nvram-wakeup -s `date -d "+15 Minutes" +%s` -A -C /etc/nvram-wakeup.conf
poweroff
```

If your computer does not wake up as expected, this may be due to a basic APM/ACPI problem, or your computer may need to run the BIOS routine once more – that is reboot – to wake up at the preset time.

Reboot not Shutdown

There is a very easy way to find out if you are facing the so-called reboot problem. Set a wakeup point for the next full quarter hour, and type *reboot* to restart your machine. Then set a wakeup time for the next full quarter hour and enter *poweroff* to power the computer off.

If the BIOS wakes up at the next full quarter hour, it has successfully read its BIOS to support wakeup.

For some motherboards the memory size accessible to */dev/nvram*, as defined in the *nvram* module, is too small to access the whole NVRAM area. Cases where *guess-helper.sh* is unable to locate the day, hour, minute, and second positions, or only finds a few fields, are indicative of this issue. The kernel sources define the memory size in *drivers/char/nvram.c*:

```
#define NVRAM_BYTES (128-NVRAM_FIRST_BYTE)
```

You need to extend the size to a full 128 bytes:

```
#define NVRAM_BYTES 128
```

This change may help *guess-helper.sh* to locate the necessary fields for your particular motherboard.

Back to the Past

settime is an ingenious method that is guaranteed to work with any motherboard. The idea is to program a fixed wakeup time in the BIOS, for example day 31 of the month at 23:59:59.

When you shutdown your machine, a script calculates the time vector between shutdown and wakeup, subtracts this vector from say July 31 2004, 23:59:59 and sets the system and RTC time to an appropriate day and time in July 2004. The next time you boot your computer, you only need to correct the time setting for everything to be as it should be.

In practical applications this approach can be a bit tricky. You have to ensure that the cor-

Listing 2: Excerpt from *settime*

```
01 #!/bin/bash
02 BiosWakeup="2004-07-31 23:59:59"
03 Wakeup=`date -d "$1" +%s`
04 Now=`date +%s`
05 Bios=`date -u -d "${BiosWakeup}" +%s`
06 Diff=$[${Wakeup}-${Now}]
07 echo "$[${Now}-${Bios}]" > /etc/timediff
08 date -u -s "${BiosWakeup} ${Diff} seconds ago" >/dev/null
09 hwclock -w --utc
```

rect time is set, whenever you boot the machine, and before *fsck* executes – this could prevent disk checks from taking place otherwise. If you correct the time before *fsck* executes, there is nowhere to make a note of the fact, as you will only have read access to the partitions.

Dual boot systems with Windows, or some other operating system, are tricky too, as they would wake up with the wrong time and date. Additionally, this method only makes sense if you boot the machine at least once every two months.

Delta T

Two separate scripts provide the answer: *settime* calculates the time vector between the actual and wakeup times, stores the difference in a file called */etc/timediff*, sets the system time to a date in July 2004, and synchronizes the RTC [3]. When you boot the system, *correcttime* is called in the first init script, reads the time vector, corrects the system time, and synchronizes the RTC. This ensures that your other start scripts will not even notice the time shift.

The *settime* method relies on the fact that the RTC continues to keep normal time. This removes the need to synchronize the system time with another system, or use an accurate clock whenever you boot. All you need to know is the number of seconds that the clock has lost in comparison to real time.

Also, there is no need to perform complicated date calculations including leap years; *date* has all the functions you require. For example,

```
date -s "+3600 seconds"
```

sets the system clock forward one hour.

To go back to the past, try the following instead:

```
date -s "3600 seconds ago"
```

Additionally, *date* can convert dates in the international standard notation, which is seconds since the epoch, as follows:

```
debian:~# date -d "2004-08-31 ➥
23:59:59" +%s
1093993199
```

The rest is easy. Listing 2 shows an excerpt from the *settime* script. Line 3 converts the wakeup time to seconds past epoch, line 4 establishes the current date in seconds past epoch, and line 6 uses these values to calculate the time vector in seconds.

Line 5 of Listing 2 calculates the wakeup time set in the BIOS, one second to midnight on July 31 2004 in this example, in seconds past epoch, allowing line 7 to calculate the difference to real time and store the value in */etc/timediff*. Now, the next time the system boots, *correcttime* just has to add the difference to the system time to obtain the real time.

Finally, lines 8 and 9 set the system and RTC time to a value in the past. As already seen in line 5, these times are converted to UTC and stored. To use local time rather than UTC for the RTC, you need to change the three lines to reflect your local timezone.

Correcting the Time

The *settime* script is only called when required, that is when you need to wakeup your machine at a preset time, and immediately before shutting down. That means *correcttime* has to find out if the RTC time is the real time, or if the computer is somewhere in the past. Listing 3 shows an excerpt from *correcttime*.

The call to *date* in line 3 discovers when */etc/timediff* was last modified, returning a value in seconds past the epoch. If the modification date in line 5 is in the future, the system time must have been set to a date in the past. Alternatively, if the modification date is in the past, the system time must be the current real time.

In lines 6 through 8, *correcttime* reads the time vector between the system and real time from the */etc/timediff* file and adds it to the current system time, thus taking the machine back to real time. The real time is then transferred to the RTC, as most distributions do an explicit sync of the system time with the time of the RTC somewhere in the boot scripts.

Listing 3: Excerpt from *correcttime*

```
01 #!/bin/bash
02 if [ -r /etc/timediff ]; then
03   Timediff=`date -r /etc/timediff +%s`
04   Now=`date +%s`
05   if [ "${Timediff}" -gt "${Now}" ]; then
06     Diff=`cat /etc/timediff | head -n 1`
07     date -s "+${Diff} seconds" >/dev/null
08     hwclock -w --noadjfile --utc
09     exit 0
10   fi
11 fi
```

Save Energy – Save Money!

A glance at your last electricity bill shows you just how expensive electricity is. You can save energy. And without suffering as a consequence if you put a bit of thought into the process. Five normal desktop PCs (150 watts each) running day and night, as is the case in many offices, will consume over 6,500 kilowatt hours, or £ 800, US$ 1,500, EUR 1,200 per year. Servers are even worse, as values of 400 watts or more per unit are quite common. A single server could consume over 3,500 kilowatt hours, that is add £ 420, US$ 780, EUR 630 to your annual electricity bill.

Shutting the computers down automatically at 9.00 pm, and booting them again at 6.00 o'clock in the morning on working days would mean a saving of more than 3,600 kilowatt hours for the desktops, and about 2,000 kilowatt hours for the server. That is equivalent to a saving of almost £ 650, US$ 1,200, EUR 1,000, not to mention the reduced environmental pollution.

If you need your computers outside of normal office hours, just press the power button. Even in a server-client-environment you can shut down both the servers and the clients – just configure the server to use "Wake on LAN" and use the *etherwake* program in the workstation init scripts. ■

INFO

[1] ACPI compatibility: *http://linvdr.org/wiki/index.php?pagename=LinVDR-Mainboards*

[2] NVRAM wakeup: *http://sourceforge.net/projects/nvram-wakeup/*

[3] The *settime* and *correcttime* scripts: *http://www.linux-magazine.com/Downloads/46/wakeup*